Generación de código de control de acceso orientado a aspectos: Un enfoque por metamodelos

Autores:

Díaz Yeregui, Pedro
Ogalla Ramírez, Jorge

Profesor director:

De Oliveira Braga, Christiano

Curso académico 2007/2008

Proyecto de Sistemas Informáticos,
Facultad de Informática,
Universidad Complutense de Madrid

ÍNDICE

RESUMEN

Presentamos una transformación basada en metamodelos desde SecureUML, un lenguaje de control de acceso a usuarios según rol, a un lenguaje de aspectos abstracto donde la política de seguridad se entiende como una instancia del metamodelo de SecureUML, y el aspecto generado se entiende como una instancia del metamodelo de aspectos. El metamodelo combinado de SecureUML y de aspectos se usa para garantizar que el aspecto generado es consistente con la política de seguridad dada. La validación de la transformación se efectúa evaluando los invariantes de los metamodelos implicados.
Hemos prototipado el enfoque como aplicación Java sobre ITP/OCL, un evaluador basado en reescritura. Retorna código validado de AspectJ desde una política de seguridad en SecureUML.

We present a metamodel-based transformation from SecureUML, a role-based access control language, to an abstract aspect language where a security policy is understood as an instance of SecureUML's metamodel and the generated aspect is understood as an instance of the aspects' metamodel. The merged metamodel of SecureUML and aspects is used to guarantee that the generated aspect is consistent with the given security policy. The validation of the transformation is done by evaluating the invariants on all involved metamodels.
We have prototyped our approach as a Java application on top of IT-P/OCL, a rewriting-based OCL evaluator. It outputs validated AspectJ code from a SecureUML policy.

AUTORIZACIÓN

Los alumnos Jorge Ogalla Ramírez y Pedro Díaz Yeregui expresamente autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales tanto la propia memoria, como el código, la documentación y el prototipo desarrollado del presente Proyecto de Sistemas Informáticos "Generación de código de control de acceso orientado a aspectos: Un enfoque por metamodelos".

Fdo:

Jorge Ogalla Ramírez

Pedro Díaz Yeregui

En Madrid a 18 de Junio del 2008.

1 Introducción

El paradigma de programación con aspectos se presenta como una solución importante para la implementación modular de requisitos *crosscutting* (G. Kiczales 1997). Requisitos *crosscutting* son aquellos cuya implementación demanda codificación en distintos puntos en una aplicación. Un ejemplo de requisito *crosscutting* son aspectos de seguridad (G. Georg 2002) (Indrakshi Ray 2003) (J. Viega 2001) (Shu Gao 2002)Este proyecto pretende ilustrar las posibilidades que ofrece, en concreto en materia de aplicación de políticas de control de acceso a usuarios. Para ello nos valdremos como ejemplo, de una aplicación desarrollada gestionar un cierto recurso en una empresa real.

1. Mostramos como el uso de aspectos puede ser eficaz a la hora de implementar un control sobre el acceso de usuarios a un recurso basándose en sus roles.

2. Mostraremos la manera de generar automáticamente el aspecto partiendo de una especificación de la política de seguridad.

3. La generación del aspecto se afronta desde el punto de vista de la transformación de metamodelos (OMG, MDA Guide Version 1.0.1 2003), combinando el metamodelo de la lenguaje de especificación de políticas de seguridad con un subconjunto básico del metamodelo de AspectJ (G. Kiczales 1997).

4. Uso de diseño por contratos (Meyer, Applying Design by Contract 1992) (Meyer, Object-Oriented Software Construction 1997) en el proceso de transformación para garantizar la conformidad con los metamodelos de entrada y salida.

5. Aplicación del proceso a un ejemplo real.

La herramienta de transformación se ha construido utilizando la herramienta ITP/OCL (Egea and Clavel 2006) para la validación de especificaciones UML con restricciones OCL (OMG, Object Constraint Language formal/06-05-01 2006). La herramienta se precarga con los metamodelos de los lenguajes utilizados: SecureUML, para la especificación de las políticas de seguridad (Basin, Doser and Lodderstedt 2002), y un subconjunto del lenguaje AspectJ.

2 Programación orientada aspectos (POA)

Desde el principio de la ingeniería de software se han producido continuos avances que han ido incrementando la capacidad de los desarrolladores de software para descomponer un sistema en módulos independientes, es lógico desde el punto de vista de la ingeniería. Lo que se quiere conseguir es proporcionar a cada uno es una función particular bien definida, esto es, facilitar la separación de intereses/competencias (concerns).

La programación con aspectos apunta hacia una separación de tareas en el contexto de la seguridad. Eso significa que el programa principal no necesita contener el código de la parte referente a la seguridad de la aplicación, se traslada a un módulo independiente. Teniendo en cuenta las bondades de la programación orientada a objetos, ésta resulta útil en las situaciones en que los conceptos que se quiere programar se pueden mapear a clases concretas. Pero en el caso concreto de la seguridad, los chequeos quedan repartidos a lo largo de todo el código y son proclives a olvidos con nefastas consecuencias.

La experiencia nos demuestra que los desarrolladores so suelen escribir buen código cuando éste debe quedar restringido por una cierta política de seguridad y cuando un error es detectado, subsanarlo suele ser enormemente complicado. Los errores están causados por dos motivos fundamentales. El primero es la escasez de herramientas y recursos para escribir aplicaciones *seguras,* como pueda ser un lenguaje específico orientado a la seguridad. La falta de generalización de éstas hace que los desarrolladores no estén acostumbrados a escribir código seguro, puesto que normalmente ni siquiera saben cuales son los riesgos. El segundo motivo esta en que ni aunque tratándose de un desarrollador experimentado, las posibilidades de un olvido o una mala utilización de los patrones de seguridad (generalmente complejos y difíciles de aplicar) son altas.

La manera en que los desarrolladores suelen trabajar es aplicar las políticas de seguridad a la aplicación ya construida y de hecho la mayor parte de las herramientas encargadas de auditar la seguridad de sistemas se basan en encontrar vulnerabilidades a los sistemas ya desarrollados, para que puedan después ser subsanados. Este método es proclive a que queden fallos sin detectar puesto que aún con una batería de pruebas de calidad, la posibilidad de que un fallo de seguridad quede sin detectar es alta.

El interés de la aplicación de la programación con aspectos a los sistemas que deban implementar políticas de seguridad está en alza y una buena prueba de ello la tenemos en (ACM, 2008) (Danish Strategic Research Council, 2008) conferencias que demuestran el interés en el sector de la informática por este campo.

Ventajas:

Los elementos referentes a la política de seguridad quedan separados del programa en sí, lo que otorga mayor claridad, mantenibilidad y reusabilidad.

La búsqueda de *seguridad por defecto* de manera que no dependa de posibles olvidos. La política se aplica de manera general y automática a lo largo de todo el código.

El código existente anterior se puede beneficiar de la aplicación de los aspectos sin ser modificado, lo que evita introducir posibles fallos.

Se pueden reutilizar aspectos en distintas aplicaciones que posean similares restricciones de seguridad y estructuras.

Los elementos que caracterizan la especificación de las políticas de acceso están en constante evolución.

Teniendo en cuenta esta necesidad de adaptabilidad y evolución, es necesario garantizar un enfoque que facilite la reutilización de los diseños y la evolución del propio soporte, es por todo esto que AOP se presenta como la más sólida alternativa, así como por su naturaleza basada en el principio de separación de tareas.

En estas dos últimas décadas el avance que más éxito ha conseguido ha sido la aparición de la *programación orientada a objetos* (POO). Este paradigma proporciona un potente mecanismo para separar intereses, pero presenta dificultades a la hora de modelar otros intereses que no pueden ser encapsulados en una única entidad o clase, ya que afectan a distintas partes del sistema. Un ejemplo sería introducir seguridad a un sistema informático. Este genera un interés (concern) que afecta a partes o a todo el sistema y genera que el sistema está atravesado (crosscutting). Este interés se engloba dentro de los intereses trasversales (crosscutting concern).

Por aclaración, los intereses transversales pueden ser desde cuestiones de alto nivel, como la seguridad o la calidad de los servicios ofrecidos, hasta cuestiones de más bajo nivel como pueden ser la sincronización, persistencia de datos, gestión de memoria, manejo de errores, logging, restricciones de tiempo, etc.

Las consecuencias directas que se generan por existencia de intereses transversales en el sistema son:

Código disperso (*scattered code*): el código que satisface una competencia transversal está esparcido por distintas partes del sistema. Facilitando que funcionalidades que tendrían que ser implementadas en un solo módulo están implementadas en varios diferentes y bloques de código duplicado en distintas partes del sistema.

Código enredado (*tangled code*): una clase o módulo además de implementar su funcionalidad principal debe ocuparse de otras competencias.

El código disperso y enredado es un código difícil de reutilizar, mantener y evolucionar. Estos efectos hacen que disminuya la calidad del software diseñado. De ahí nace la necesidad de nuevas técnicas que nos ayuden a conseguir una separación de intereses clara, para así reducir la complejidad

del sistema a implementar y mejorar aspectos de calidad como la adaptabilidad, extensibilidad, mantenibilidad y reutilización.

En los últimos años han ido surgiendo distintas técnicas o paradigmas que intentan solucionar el problema de modularizar las competencias transversales. La más extendida actualmente es la ***programación orientada a aspectos*** (POA).

La POA es un nuevo paradigma de programación que, situándose por un nivel de abstracción por encima, permite capturar los intereses transversales en entidades bien definidas llamadas **aspectos**, consiguiendo una clara separación de intereses, eliminando así el código disperso y enredado y los efectos negativos que este supone.

2.1 Fundamentos de la POA

Para establecer las bases de la POA, se pueden agrupar los lenguajes procedurales y los funcionales formaban parte de una misma familia de lenguajes, los **lenguajes de procedimiento generalizado** (*generalizad-procedure languages*), ya que sus mecanismos de abstracción y composición tienen una raíz común en forma de un procedimiento generalizado. Para los lenguajes OO el procedimiento generalizado es una clase, para la programación funcional una función, para la programación procedural un procedimiento. Gregor Kiczales clasifica las propiedades a implementar usando lenguajes de procedimiento generalizado en dos tipos:

Un componente: puede encapsularse claramente dentro de un procedimiento generalizado. Los componentes son unidades de descomposición funcional del sistema.

Un aspecto: no puede encapsularse claramente en un procedimiento generalizado. Suelen ser propiedades que afectan al rendimiento o a la semántica de los componentes.

Gregor Kiczales enuncia como objetivo principal de la POA: "*proporcionar un marco de trabajo que permita al programador separar claramente componentes y aspectos a través de mecanismos que hagan posible abstraerlos y componerlos para producir el sistema global.".*

También proporciona una definición formal del término aspecto: "*un aspecto es una unidad modular que se dispersa por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de implementación es una unidad modular del programa que aparece en otras unidades modulares del programa*."

Podemos hablar de los aspectos como aquellas propiedades, conceptos o intereses que atraviesan el sistema (*crosscutting concern*), no pudiendo ser encapsulados en una única entidad y dispersándose por todo el código. Los aspectos son la unidad básica de la POA. Ejemplos de aspectos son los

relacionados con la gestión de memoria, sincronización, manejo de errores, seguridad, etc.

Volviendo a nuestro ejemplo principal, es decir, la gestión de la seguridad de un sistema informático, todo el posible código de gestión de la seguridad se entremezclaría con el que implementa la funcionalidad de la aplicación, dando lugar a un código disperso y enredado.

En cambio, a lo largo de esta memoria podremos concluir que usando en la implementación mediante POA de un modelo de seguridad SecureUML, se consigue una separación de intereses completa, consiguiendo encapsular todas las restricciones de seguridad que se imponen en el sistema dado el modelo SecureUML a aplicar, en un aspecto de control de acceso.

Se puede concluir una gran mejora en modularidad al estar todas las restricciones de seguridad en el aspecto. Cambios en la seguridad y sus restricciones solo implicarían cambios en el aspecto y en ningún modulo más del sistema.

2.2 Diseño orientado aspectos

De acuerdo con (G. Kiczales 1997)**, las fases del diseño orientado a aspectos son:**

Etapa 1: Identificar competencias/intereses

Consiste en descomponer los requisitos del sistema en competencias y clasificarlas en:

Competencias básicas (*core-concern*): las que están relacionadas con la funcionalidad básica del sistema, de carácter funcional. Son los *componentes* mencionados anteriormente.

Competencias transversales (*crosscutting-concern*): las que afectan a varias partes del sistema, relacionadas con requerimientos no funcionales del sistema, normalmente de carácter no funcional. Son *aspectos* mencionados anteriormente.

Etapa 2: Implementar competencias/intereses

Consiste en implementar cada interés independientemente:

Los componentes a través de POO o programación procedural o programación funcional. El lenguaje se utiliza en este caso es el denominado **lenguaje base** en este proyecto será Java.

Para implementar las competencias transversales usaremos **lenguajes orientados a aspectos**, encapsulando cada competencia en unidades llamadas aspectos. Estos lenguajes orientados a aspectos deben ser compatibles con el lenguaje base para que los aspectos puedan ser combinados con el código que implementa la funcionalidad básica y así obtener el código final. Normalmente los lenguajes orientados a aspectos

suelen ser extensiones del lenguaje base, como es el caso de AspectJ, que es el que utilizaremos en este proyecto, o independientes.

Etapa 3: Componer el sistema final

Este proceso se conoce como **entretejido** (*weaving*). Este combina los aspectos con los módulos que implementan la funcionalidad básica del sistema dando lugar al sistema final. El modulo encargado de realizar este proceso recibe el nombre de **tejedor de aspectos** (*weaver*).

En si, las acciones de entretejido son:

En encontrar los puntos de enlace (Joint points) en el código del lenguaje fuente. Estos son puntos específicos en el código fuente.

Si estos puntos de enlace se ven afectado por algún aspecto, se aplican reglas de entretejido que introducen el código fuente especificado en el/los punto/s de corte (pointcuts) que capturan el punto de enlace y el posible código fuente especificado en el/los advice/s sobre el/los punto/s de corte (pointcuts).

Concretando para nuestro proyecto, el tejedor de aspectos de AspectJ es estático. El entretejido tiene lugar a nivel de byte-code, es decir, el compilador de AspectJ entreteje los aspectos en los ficheros .class de nuestra aplicación, generando los .class de salida correspondientes. La aplicación se ejecuta al final como una aplicación Java.

2.3 AspectJ

AspectJ es un lenguaje orientado a aspectos de propósito general, creado en 1998 por *Gregor Kiczales* y el grupo de investigación que dirige, el *Xerox PARC*. En diciembre del 2002, el proyecto AspectJ fue cedido por sus creadores a la comunidad open-source *Eclipse.org*.

Es una extensión de Java para soportar la definición y manejo de aspectos. Además, el compilador de AspectJ genera byte-codes compatibles con cualquier máquina virtual de Java. Implementar AspectJ como una extensión de Java hace que el lenguaje sea fácil de aprender, ya que utiliza elementos comunes con Java. La única dificultad viene dada por el aprendizaje del paradigma de aspectos y la sintaxis especifica de AspectJ.

AspectJ es un lenguaje relativamente nuevo y por lo tanto en continua evolución.

En el momento de realizar este proyecto, la última versión estable era AspectJ 1.5.4, que es la que se ha utilizado para el aprendizaje y desarrollo de este trabajo fin de carrera.

2.3.1 Especificación de AspectJ

En AspectJ, para expresar la funcionalidad de los aspectos se utiliza Java, mientras que para especificar las reglas de entretejido (*weaving rules*) el lenguaje proporciona una serie de constructores, que se detallan a continuación. La implementación de las reglas de entretejido se suele llamar

crosscutting (los aspectos "atraviesan" las clases existentes). Existen dos tipos de *crosscutting*: *dinámico* y *estático*. Para evaluar condiciones de seguridad (permisos), que es nuestro interés, solo necesitaremos crosscutting dinámico. Únicamente vamos a explicar este tipo.

Crosscutting dinámico: los aspectos afectan al comportamiento de la aplicación, modificando o añadiendo nuevo comportamiento. Los constructores:

Puntos de enlace (*join point*): son puntos bien definidos en la ejecución de un programa. Ejemplos son: llamadas a métodos, lectura de un atributo, escritura de un atributo, etc. Representan los lugares donde los aspectos pueden añadir su comportamiento.

Puntos de corte (*pointcut*): agrupan puntos de enlace y permiten exponer su contexto a los avisos a través de parámetros. Por ejemplo, mediante un punto de corte podemos agrupar las llamadas a todos los métodos de cierta clase (agrupando la captura de los Joint point de la llamada a dichos métodos).

Avisos (*advice*): especifican el código que se ejecutará en los puntos de enlace que satisfacen cierto punto de corte. Hay distintos tipos, que hacen que se pueda ejecutar funcionalidad antes, después o sustituir la funcionalidad completamente por otra alternativa.

Por aclaración, los puntos de corte indican *dónde* y los avisos *qué hacer*.

Un ejemplo fácil de entender. Para la seguridad y control de acceso, si queremos ver que un usuario tiene permisos para ejecutar un método, se hará lo siguiente. El aspecto de control de acceso (seguridad) capturará cada método con un punto de corte. A cada punto de corte se le asociará un aviso que realice la comprobación de permisos para dejar al usuario ejecutar el método (si tiene permiso) o lanzar una excepción de error.

Los puntos de corte y avisos se encapsulan en un entidad llamada **aspecto** (*aspect*). Un aspecto es la unidad básica de AspectJ. Además, éste también puede contener atributos y métodos como una clase Java normal.

2.3.1.1 Aspecto

Para encapsular el interés transversal en una entidad (aspecto) primero se procede a identificar aquellos puntos de enlace donde queremos añadir el comportamiento cruzado, agrupándolos con un punto de corte. Posteriormente, se implementa el comportamiento transversal con un aviso, cuyo cuerpo será ejecutado cuando se alcance alguno de los puntos de enlace que satisfacen el punto de corte definido en el paso anterior.

Vamos a aplicar los pasos descritos anteriormente para implementar una competencia transversal con AspectJ. Vamos a definir el problema.

Se presentan dos entidades: **Client** y **Resource**.

Client contiene a clientes que van a pedir el recurso secreto (método getExtResource()) y además es la clase principal.

```
public class Client {

   private static String acceso1 = "usuario1";

   private static String acceso2 = "root";

   public static void main(String[] args) {

      Client cli = new Client();

      //Solicitud del recurso por la clase Client como acceso1

      try {

            System.out.println(cli.getExtResource(acceso1));

      } catch (Exception e) {

            System.out.println("No tienes permisos suficientes para
            acceder al recurso");

      }

      //Solicitud de recurso por la clase Client como acceso2

      try {

            System.out.println(cli.getExtResource(acceso2));

      } catch (Exception e) {

            System.out.println("No permisos suficientes para acceder
            al recurso");

      }

}

      String getExtResource (String usr)throws Exception{

            Resource r = new Resource();

            return r.dameRsc(usr);

      }

}
```

Resource contiene un recurso, un método para obtenerlo y otro método que verifica si un usuario tiene permisos de acceso a ese recurso.

```
public class Resource {

      private String dato = "Recurso secreto";

      private String acceso1 = "admin";

      private String acceso2 = "root";

      public String dameRsc (String user) throws Exception{
```

```
            return dato+" obtenido por "+user;

      }

      public boolean isGranted(String user){

            if (user.equalsIgnoreCase(acceso1) ||

                user.equalsIgnoreCase(acceso2)

                ){

                  return true;

            }

            return false;

      }

}
```

Hasta este punto hemos codificado la funcionalidad base. Hasta aquí, si la aplicación se ejecuta, cualquier usuario podrá obtener el recurso.

Ahora implementamos la seguridad global del sistema, que es una competencia transversal, y será llevada a cabo por un aspecto. El aspecto captura el joint point **`dameRsc()`** en el pointcut **`callOndameRsc`**. Después, utiliza un advice sobre el pointcut para implementar el control de que el usuario tiene permiso.

El método **`isGranted()`** devuelve **`true`** si el usuario que tiene por atributo tiene permisos para obtener el recurso y **`false`** si no los tiene. Si no tiene permisos se lanza una excepción, así nunca se devolverá el recurso al cliente. Si tiene permisos se ejecutará correctamente el método **`dameRsc()`** y el cliente obtendrá el recurso.

```
public aspect SecurityAspect {

pointcut callOndameRsc(Resource res,String param): call (String
Resource.dameRsc(String)) && args(param)&& target(res);

before(Resource res,String param) throws Exception :
callOndameRsc(res,param){

      System.out.println("Comienzo de la validación");

      if (res.isGranted(param)){

            System.out.println("validación correcta");

      }

      else{

            System.out.println("validación incorrecta");

            throw new Exception( "fallo en validación");

      }
```

```
 }

}
```

Una prueba de ejecución de la aplicación cuando el usuario con nombre *usuario1* intenta obtener el recurso que está limitado para *root* y *admin* daría como resultado:

```
Comienzo de la validación

validación incorrecta

No tienes permisos suficientes para acceder al recurso
```

El usuario con nombre root obtiene el recurso:

```
Comienzo de la validación

validación correcta

Recurso secreto obtenido por root
```

Por lo tanto, usando la POA logramos encapsular el comportamiento cruzado del control de permisos sobre el sistema, logrando una separación de intereses completa. Gracias a esta separación cualquier modificación de políticas de permisos se hará solamente en el aspecto. Se observa la gran modularidad que se ha conseguido. Además ganamos en adaptabilidad, extensibilidad, mantenibilidad y reutilización.

Visto el ejemplo vamos a entrar en profundidad en la parte de AspectJ que más nos interesa.

2.3.1.2 Puntos de enlace

Los puntos de enlace son puntos bien definidos en la ejecución de un programa. Son puntos sobre el flujo de ejecución de un programa.

AspectJ soporta los siguientes tipos de puntos de enlace:

- **Llamada a un método**: ocurre cuando un método es invocado por un objeto.
- **Llamada a un constructor**: ocurre cuando un constructor es invocado durante la creación de un nuevo objeto.
- **Ejecución de un método**: ocurre cuando se ejecuta un método y engloba la ejecución del cuerpo de dicho método.
- **Ejecución de un constructor**: abarca la ejecución del cuerpo de un constructor durante la creación de un nuevo objeto.
- **Lectura de un atributo:** ocurre cuando se lee un atributo de un objeto dentro de una expresión.
- **Escritura de atributo**: ocurre cuando se asigna un valor a un atributo de un objeto.

Ejecución de un manejador de excepciones: ocurre cuando un manejador es ejecutado, es decir, cuando se ejecuta el cuerpo del `catch()`.

Ejecución de un aviso: comprende la ejecución del cuerpo de un aviso.

Una vez aclarados los puntos de enlace más comunes y sus particularidades, lo siguiente es como identificador mediante los puntos de corte.

2.3.1.3 Puntos de corte

Los puntos de corte identifican la captura una serie de puntos de enlace y permiten exponer su contexto a los avisos (*advice*). Son las unidades de un determinado aspecto, que capturar puntos de enlaces para dicho aspecto y se tratan con avisos sobre ellos.

Usando las construcciones ofrecidas por el lenguaje, podremos crear puntos de corte desde muy generales, por ejemplo que capturen todos los puntos de enlace del sistema, o muy específicos, que identifican a un único punto de enlace.

Existen puntos de corte **anónimos** y con **nombre**. Para nuestro proyecto solo nos interesan los identificados con un nombre.

La sintaxis de un punto de corte es la siguiente:

<pointcut> ::= <access_type> [**abstract**] **pointcut**

<pointcut_name>({<parameters>}) **:** {[**!**] designator [**&&** | **||**]};

designator ::= designator_identifier(<signature>)

Un punto de corte con nombre consta del modificador de acceso (access_type), por defecto package, la palabra reservada pointcut seguida del nombre del punto de corte y de sus parámetros, que permiten exponer el contexto de los puntos de enlace a los avisos (*advice*). Por último, vienen las expresiones que identifican los puntos de enlace precedidas del carácter dos puntos "**:**".

Ejemplo:

```
pointcut foo(Point p): (call( * Point.set*(int) ) || call (* point. get *())) && target(p);
```

La sintaxis de los distintos tipos de signatura que pueden aparecer en un punto de corte son:

Signatura de un método:

<access_type> <ret_val> <class_type>.<method_name>(<parameters>)

[**throws** <exception>]

Ejemplo:

public void Point.setX(int)

Signatura de un constructor:

<access_type> <class_type>**.new**(<parameters>)[**throws** <exception>]

Ejemplo:

public Point.new(int,int)

Signatura atributo

<access_type> <field_type> <class_type>.<field_name>

Ejemplos de signatura de atributo son:

public int Point.x

Signatura tipo

<type>

Los tipos en AspectJ son clase, interfaz, tipo primitivo o incluso un aspecto.

Por lo tanto ejemplos de signatura de tipo podrían ser:

Point

Exception

Comodines son utilizados para identificar puntos de enlace que tienen características comunes:

***** : Cualquier número de caracteres excepto el punto y en otros casos representa cualquier tipo (clase, interfaz, tipo primitivo o aspecto).

.. : Cualquier número de caracteres, incluido el punto. Cuando se usa para indicar los parámetros de un método, significa que el método puede tener un número y tipo de parámetros arbitrario.

Las expresiones que identifican los puntos de enlace se pueden **combinar** mediante el uso de los siguientes operadores lógicos:

exp1* | | *exp2 : operador lógico OR.

e*xp1* && *exp2*: operador lógico AND.

! *exp* : operador de negación.

La precedencia de estos operadores lógicos es la misma que en Java.

AspectJ proporciona una serie de descriptores de puntos de enlace que nos permiten agruparlos según diferentes criterios. Los que nos interesan son:

Basados en las categorías de puntos de enlace: capturan los puntos de enlace según la categoría a la que pertenecen: llamada a método (call), ejecución de método (execute), lectura de atributo (get).

```
call (String Resource.dameRsc(String))
```

Basados en los objetos en tiempo de ejecución: capturan los puntos de enlace cuyo objeto actual (this) u objeto destino (target) son de un cierto tipo. Nos permiten obtener el contexto del punto de enlace.

```
target(res)
```

Basados en los argumentos del punto de enlace: capturan los puntos de enlace cuyos argumentos son de un cierto tipo mediante el descriptor args. También puede ser usado para exponer el contexto.

```
args(param)
```

Mostramos un ejemplo:

```
pointcut callOndameRsc(Resource res,String param): call (String Resource.dameRsc(String)) && args(param)&& target(res);
```

El pointcut captura el jointpoint *Resource.dameRsc(String*), además se tiene que cumplir que el parámetro *param* sea de tipo *String*. El pointcut recibe el contexto del objeto sobre el que se realiza la llamada a *dameRsc(String)* y el parámetro de esa llamada.

2.3.1.4 Avisos

Los avisos son construcciones que especifican las acciones a realizar en los puntos de enlace que satisfacen cierto punto de corte. Mientras los puntos de corte indican *donde*, los avisos especifican *que hacer*.

Un aviso lo podemos dividir en tres partes: declaración, especificación del punto de corte y cuerpo del aviso.

El cuerpo del aviso se ejecuta antes del punto de enlace capturado. Si se produjera una excepción durante la ejecución del aviso, el punto de enlace capturado no se ejecutaría. Se utilizara constantemente en la comprobación de permisos en nuestro proyecto (precondiciones).

Ejemplo anterior de comprobación de permisos para obtener el recurso:

```
before(Resource res,String param) throws Exception : callOndameRsc(res,param){
        System.out.println("Comienzo de la validación");
        if (res.isGranted(param)){
                System.out.println("validación correcta");
        }
        else{
                System.out.println("validación incorrecta");
                throw new Exception( "fallo en validación");
        }
```

```
}
```

2.3.1.5 Avisos *after*

El cuerpo del aviso se ejecuta después del punto de enlace capturado. Mencionamos los tres tipos de avisos *after*:

- Aviso ***after*** (no calificado): el aviso se ejecutará siempre, sin importar si el punto de enlace finalizó normalmente o con el lanzamiento de alguna excepción. Se comporta como un bloque finally en Java.

- Aviso **after returning**: el aviso sólo se ejecutará si el punto de enlace termina normalmente y si el tipo del valor retornado por el punto de enlace capturado es compatible con el indicado en la cláusula returning. Cabe destacar que se puede acceder al valor de retorno devuelto por el punto de enlace.

- Aviso **after throwing**: el aviso sólo se ejecutará si el punto de enlace finaliza con el lanzamiento de una excepción, pudiendo acceder a la excepción lanzada. Al igual que ocurría con los avisos *after returning*, un aviso *afterthrowing* sólo se ejecutará si el tipo de la excepción lanzada por el punto de enlace capturado coincide con el indicado en la cláusula throwing.

2.3.1.6 Avisos around

Se ejecutan en lugar del punto de enlace capturado, ni antes ni después. Podemos remplazar la ejecución, ignorarla, cambiar su contexto…

Para ejecutar el punto de enlace original dentro del cuerpo del aviso usamos un el método especial: **proceed()**.

La declaración de un aviso está delimitada por el carácter "**:**" y consta del tipo de retorno (sólo para avisos de tipo **around**), el tipo de aviso (**before**, **after** o **around**) y su lista de parámetros, que expone información de contexto para que pueda ser usada en el cuerpo del aviso. Después de la lista de parámetros se especifican las excepciones que puede lanzar el cuerpo del aviso mediante la cláusula throws. Posteriormente, ":" y el nombre del pointcut. Por último, el cuerpo de un aviso encierra las acciones a realizar. Un ejemplo de un aviso es:

```
before(Resource res,String param) throws Exception : callOndameRsc(res,param){… }
```

Mediante los **parámetros** de un aviso se puede exponer información de contexto para que sea accesible desde el cuerpo de un aviso. Para ligar estos parámetros con la información de contexto se utiliza:

- El descriptor de punto de corte args() para exponer los argumentos de un punto de enlace.
- El descriptor de punto de corte this() para obtener el objeto actual asociado a la ejecución de un punto de enlace.
- El descriptor de punto de corte target() para exponer el objeto destino asociado a la ejecución de un punto de corte.

- Las cláusulas returning y throwing para recuperar el valor retornado o la excepción lanzada por un punto de enlace. Sólo aplicables a los avisos de tipo around.

Una mención importante es que, a la hora de utilizar parámetros, tanto en los puntos de corte como en los avisos, se tiene que cumplir la siguiente regla: *todos los parámetros que aparecen a la izquierda del carácter delimitador dos puntos ":" deben estar ligados a alguna información de contexto a la derecha de los dos puntos.*

También podemos acceder al contexto de un punto de enlace de **forma reflexiva**, mediante una serie de variables especiales proporcionadas por AspectJ:

thisJoinPoint: variable de tipo org.aspectj.lang.JoinPoint que nos permite acceder a información, tanto estática como dinámica, sobre el contexto del punto de enlace actual mediante una serie de métodos, en esta memoria se omiten su explicación. En la bibliografía (G. Kiczales, 1997) se pueden obtener la información completa de dichos métodos.

thisJoinPointStaticPart: variable perteneciente a la clase org.aspectj.lang.JoinPoint.StaticPart con la que sólo podemos acceder a las partes estáticas del contexto de un punto de enlace.

Auque podemos obtener a través del acceso reflexivo todo el contexto, se aconseja mejor obtener el contexto por parámetros y utilizando args(), this(), y target()ya que este tipo de acceso es muy lento y menos metódico.

3 Control de Acceso a usuarios Basado en Rol (RBAC)

Control de acceso a usuarios basado en acceso (en adelante RBAC) es el método para regular el acceso a un sistema o a un recurso en base al rol del usuario dentro de una jerarquía. En éste contexto, acceso es la habilidad de un usuario a efectuar una determinada operación, ya sea visualizar, crear o modificar el recurso. Los roles se definen conforme a las competencias y responsabilidades en la cadena jerárquica.

Correctamente implementado RBAC permite a los usuarios llevar a cabo las acciones que tiene autorizadas, restringiéndolas en base a las normas especificadas de una manera dinámica. Esto contrasta con antiguos modelos de control de acceso donde las restricciones quedaban especificadas de manera permanente para los distintos roles.

Un hecho importante en RBAC ha sido la patronización propuesta por la National Institute of Standards and Techonlogy (NIST) (Role Based Access Control 2004). El estándar se basó en trabajos previos propuestos por el ACM (Sandhu, Ferraiolo and Kuhn 2000) (Jaeger and Tidswell 2000) y las propias investigaciones del NIST acerca de cómo estaba siendo implementada esta característica en la industria.

No todas las aplicaciones requieren una misma política de control de acceso y por lo tanto éstas mantendrán una serie de características que las diferenciarán. Es por esto que el modelo de referencia de RBAC es dividido en una serie de capas que complementan a la estructura básica o núcleo.

En el caso que nos ocupa, la estructura de la política de acceso de usuarios necesitará del conjunto básico de reglas de RBAC, el *core RBAC,* y el componente *Hierarchical RBAC,* que añade relaciones para dar soporte a jerarquías entre roles, introduciendo, además, el concepto de conjunto de permisos dados a usuarios en un rol.

Comencemos con las definiciones básicas de los elementos partícipes:

- Un *usuario* se definirá como una persona, aunque el concepto se pueda extender a una máquina, una red o un agente autónomo. Un *rol* es una funcionalidad dentro del contexto de una organización con una semántica asociada en lo que respecta a su autoridad y a la responsabilidad que se le confiere al *usuario* que toma el *rol.*

- Un *permiso* es la aprobación para realizar una operación en uno o mas objetos protegidos por la política RBAC.

- Una *operación* es la representación física de un programa que en su invocación ejecuta alguna función para el usuario. Los tipos de operaciones que RBAC controlará dependerán del dominio de aplicación del sistema.

El objetivo final del RBAC es proteger recursos de un sistema y éstos pueden ser de muy diversa naturaleza. El conjunto de objetos que cubrirá incluye todos aquellos que aparezcan en los permisos que estarán asignados a roles.

El concepto de relaciones entre roles es fundamental. Un usuario puede tener asignados uno o más roles y un rol puede tener asignados uno o más usuarios. Esta configuración ofrece una gran flexibilidad y granularidad a las asignaciones entre roles y usuarios. Esta flexibilidad permite que se haga un uso mas restringido de los permisos ya que, de otra forma, se corre el riesgo de otorgar permisos innecesarios a ciertos usuarios. Como se puede ver en la figura Figure 3-1

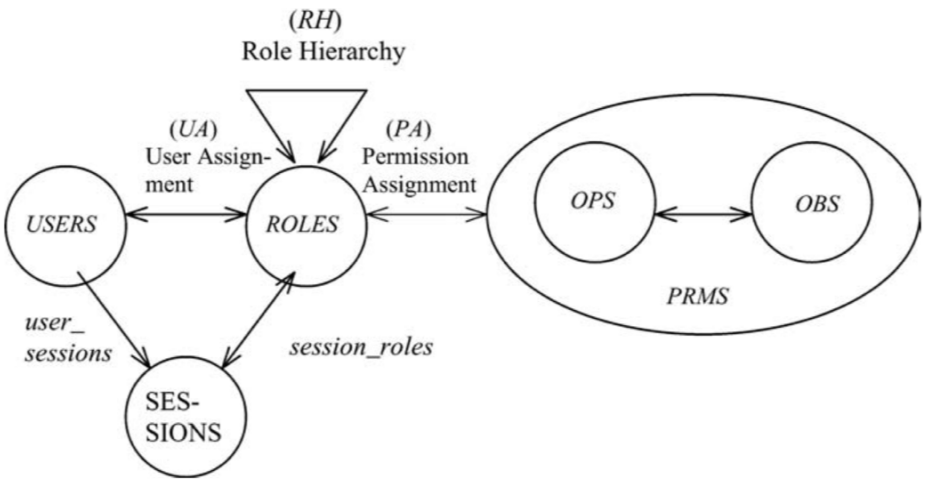

Figure 3-1

El componente *Hierarchical RBAC* introduce jerarquías entre los roles y esta suele ser una de las características mas demandadas, puesto que dan una estructura natural a los roles que reflejan las líneas de autoridad y responsabilidad de las organizaciones. Las jerarquías entre roles definen una relación de herencia entre ellos. Esta herencia se aplica al concepto de permisos: el rol r1 “hereda” del rol r2 si todos los permisos de r2 son también permisos de r1.

En nuestro ejemplo en concreto la noción de *sesión* no se ha utilizado a pesar de que este concepto queda también contemplado en el estándar.

4 Model Driven Architecture (MDA)

En el campo del desarrollo de software, podemos hablar de un modelo de manera análoga a como nos referimos a los planos de un edificio en su etapa de construcción. Nos permiten ver la estructura de la aplicación eliminando los detalles que nos impidan ver el marco general. En ésta misma línea de pensamiento un metamodelo será la estructura común sobre la que todas las instancias del mismo (en éste caso modelos) estén construidas

Entendamos Model Driven Architecture (OMG, MDA Guide Version 1.0.1 2003), en adelante (MDA), como un paso mas en el campo del desarrollo de software.

La OMG surgió ante la necesidad de simplificar el proceso de construcción de software, y, para ello, se basa en la MDA para guiar a la industria hacia un software portable, reutilizable, construido por componentes y que se base en los modelos que se entiendan como estándar en ese momento.

La arquitectura guiada por modelos (MDA) comienza con la idea de separar la especificación de un sistema de los detalles de como el sistema usa su *Plataforma*, entendiendo plataforma como un conjunto coherente de funcionalidades a través de interfaces y patrones de uso que cualquier aplicación soportada por esta plataforma pueda usar, abstrayéndose de los detalles de como esa funcionalidad es implementada. En resumen:

Utilidad:

> Especificar un sistema independientemente de la plataforma que lo soporta.
>
> Especificar plataformas
>
> Escoger una plataforma en particular para el sistema
>
> Transformar la especificación del sistema en una instancia en particular para una plataforma.

Objetivo:

> Portabilidad, Interoperatividad, reusabilidad

Tomando la definición de modelo de un sistema como la descripción de la especificación de ese sistema y su entorno para un determinado propósito, la arquitectura guiada por modelado será una manera de desarrollar sistemas, que incrementa el poder de los modelos en trabajo y será guiada por modelado puesto que permite usar los modelos para dirigir el proceso.

Durante el proceso de creación de software, la idea de tener separadas la especificación del programa y la implementación es, en sí, lo que se ha denominado MDA por la OMG. De esta forma, la arquitectura de un determinado programa queda separada en tres niveles de abstracción:

- Computational Independent Model (Modelo Independiente de la Computación);

- Plataform Independent Model (Modelo Independiente de la Plataforma);
- Plataform Specific Model (Modelo Específico de la Plataforma).

El primer nivel, CIM, es una vista del sistema eliminando los detalles de la estructura del sistema (a veces denominado Modelo del Dominio) pero no entraremos en detalles sobre él puesto que resulta irrelevante en nuestro proyecto ya que suponemos que tenemos como punto de partida un sistema ya modelado. El segundo nivel, PIM, permite centrarse en la especificación (análisis/diseño) del software, olvidándose del resto de detalles referentes a la plataforma en la que se implemente. Como contraposición, el último nivel será el PSM que combina las especificaciones del PIM con los detalles específicos de la plataforma donde se va a utilizar el sistema.

Los niveles MDA se componen de muchos elementos y sus relaciones. Esto aumenta la dificultad para tratar temas que involucren a todo el programa de una manera independiente. El problema ocurre cuando ciertas entelequias del programa no pueden ser modelados como entidades propias. De esta forma, esa funcionalidad queda dispersa (*scattered*) por todo el conjunto de entidades que la conforman y éstas están además enredadas entre sí puesto que modelan varias características. Es aquí donde podremos comprobar la eficacia de la Aspect Oriented Programming (AOP) .

La separación de asuntos desde el punto de vista de modelado de software ha sido objeto de numerosos estudios (Jacobson, Use Cases and Aspects - Working Seamlessly Together, 2003) (Kulkarni & Reddy, Separation of concerns in Model Driven Development) (Kulkarni & Reddy, Integrating aspects with Model Driven Software Development, 2003)

4.1 Combinación disjunta de metamodelos

El enfoque que presenta OMG para la combinación de metamodelos pretende únicamente ser sugestivo, en el sentido de que deberá ser estudiado para cada caso en particular, puesto que el mapeo de elementos así como la intersección de ellos serán específicos de cada combinación. Podemos hablar de "mezclado" de modelos en el sentido en que es un concepto parecido al "weaving" del que hablábamos en AOP. En la imagen Figure 4-1 vemos como se genera una plataforma específica del modelo al mezclar la plataforma independiente con un modelo concreto.

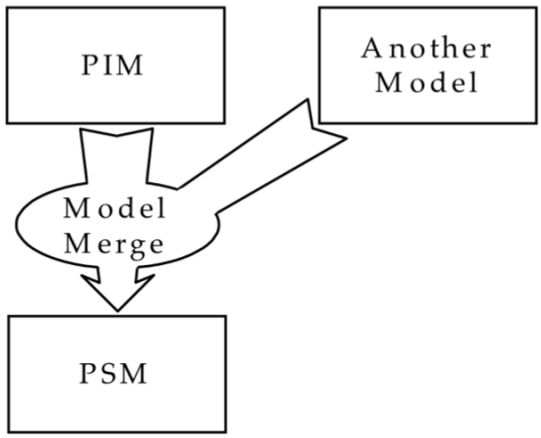

Figure 4-1

4.2 Transformación de Metamodelos

El proceso de transformación descrito en la Figure 4-2 ilustra un enfoque de transformación de metamodelos en el que se prepara un modelo concreto usando un lenguaje independiente de plataforma que a su vez es especificado por un metamodelo. Se toma una plataforma concreta. La especificación de la transformación para esta plataforma se prepara de la misma manera que en el proceso de combinación de metamodelos, esta especificación de la transformación se hace en términos de mapeo entre ambos metamodelos. Éste mapeo será el que guíe la transformación del PIM para producir el PSM

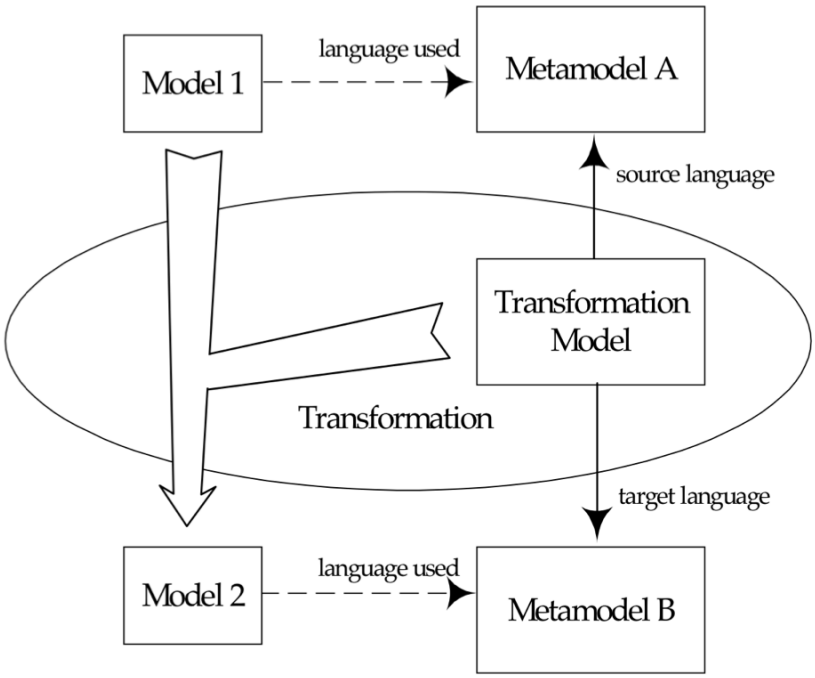
Model 1
language used
Metamodel A
source language
Transformation Model
Transformation
target language
Model 2
language used
Metamodel B

Figure 4-2

5 Metamodelos utilizados en la transformación

5.1 Metamodelo SecureUML

El lenguaje de modelado hemos utilizado en nuestra implementación del transformador, ha sido SecureUML que define un vocabulario para anotar modelos basados en UML con la información relevante al control de acceso. Se basa en el modelo explicado con anterioridad de RBAC, con la salvedad ya expuesta de que no se tratan las sesiones y con el añadido del soporte para la especificación de restricciones de autorización.

La estructura de SecureUML cumple con las especificaciones de lo que el OMG entiende como arquitectura para desarrollo de software guiado por modelos.

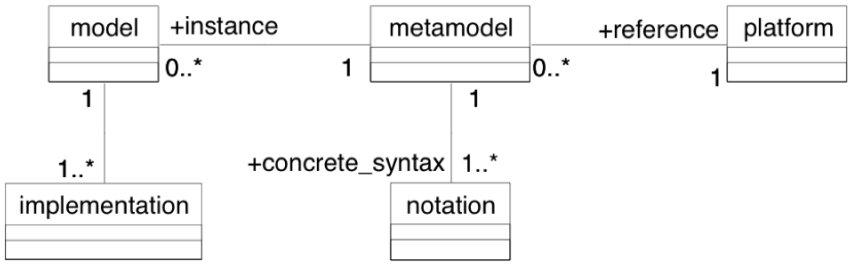

Figure 5-1

Un modelo es un sistema software abstraído a un cierto nivel. Un metamodelo define la sintaxis de una clase de modelos. Cada metamodelo tiene asignada una plataforma particular, que es su plataforma de referencia.

SecureUML define un vocabulario para expresar las características del control de accesos como los roles, los permisos asignados a los roles y las relaciones entre usuarios y roles. El enfoque se basa en que este lenguaje se use complementando a otro lenguaje de modelado. De ésta forma se podrán especificar sistemas a distintos niveles de abstracción utilizando la misma notación.

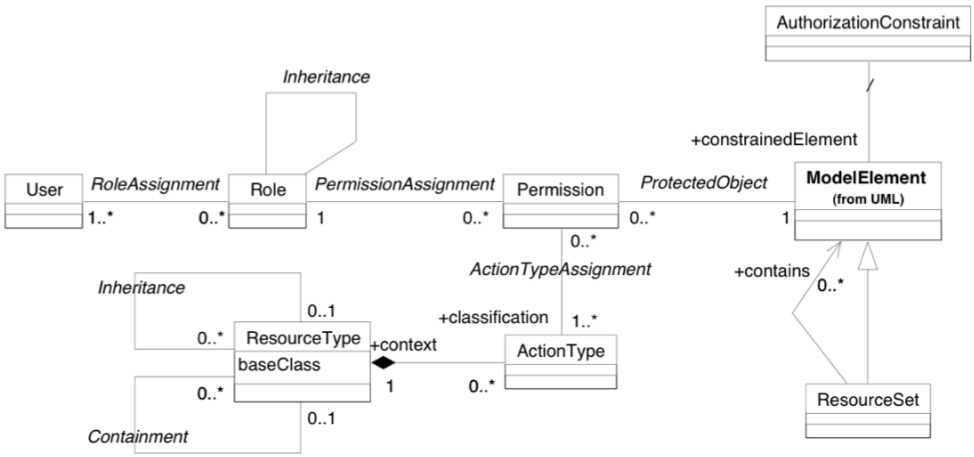

Figure 5-2

El metamodelo se define como una extensión del metamodelo de UML. Los conceptos de RBAC se representan directamente como tipos del metamodelo. Introducen los tipos Usuario, Rol y Permiso, así como las relaciones entre ellos.

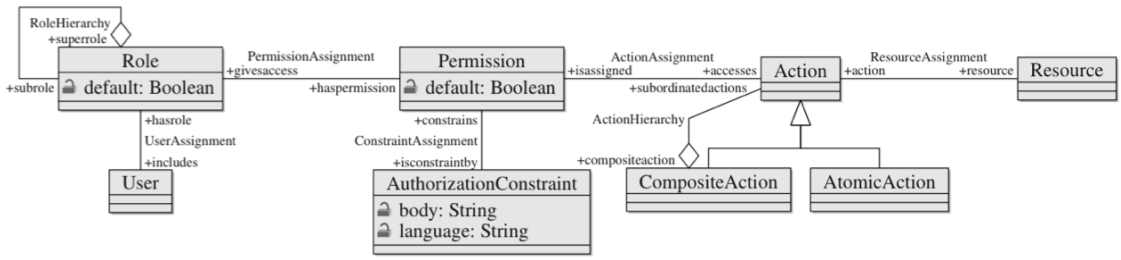

Figure 5-3

Dada su naturaleza global los recursos protegidos se representan no como una clase sino como un metamodelo, donde cada modelo UML pueda tomar parte como recurso protegido. Adicionalmente dotan al modelo de el tipo *ResourceSet* que representa a un conjunto de elementos de modelo usados para definir los permisos o las restricciones.

Un Permiso es una relación entre un rol y un *ModelElement* o un *ResourceSet* . La semántica de un permiso se define mediante un elemento del tipo *ActionType*, que representan una clase de operaciones sobre un determinado recurso protegido, como puedan ser los métodos accesotes y mutadores de un atributo, o un método que involucre algún tipo de operación que deba ser controlada desde el punto de vista de la seguridad. Una clase puede contener métodos y atributos y podremos adjuntar un permiso a la

clase con una acción del tipo *read*, lo que representaría el permiso para todos los métodos sin efectos colaterales que lean cualquier atributo de la clase.

El conjunto de tipos de acciones puede ser definido con mayor libertad usando los elementos de tipo *ResourceType*. Un *ResourceType* define todos los tipos de acciones posibles para un determinado metamodelo. La conexión con el metamodelo se representa mediante el atributo *baseClass* que condene el nombre del tipo o del estereotipo. El conjunto de tipos de recurso, sus acciones y la definición de su semántica en una plataforma determinada definen el tipo *recurso* para una plataforma.

Un *AuthorizationConstraint* es una parte de la política de control de acceso de una aplicación, expresa las precondiciones impuestas en cada llamada a una operación sobre un recurso, que generalmente dependerá del estado del recurso, la llamada en curso o el entorno. Se deriva de tipo de UML *Constraint* como tal se asigna directa o indirectamente a un elemento del modelo que represente un recurso protegido.

Como ya hemos explicado el lenguaje SecureUML proporciona la capacidad de especificar políticas de seguridad para acciones sobre recursos protegidos. Sin embargo no define cual será el recurso protegido y que acciones podrá el cliente realizar sobre él. Éstas se especifican en un *dialecto* que dependerá del lenguaje de modelado que estemos utilizando. En nuestro caso en particular se trata de ComponentUML que proporciona un subconjunto de UML, de tal forma que *Entity* se relaciona con se relaciona con *Association* y tienen *Attributes* y/o *Methods.*

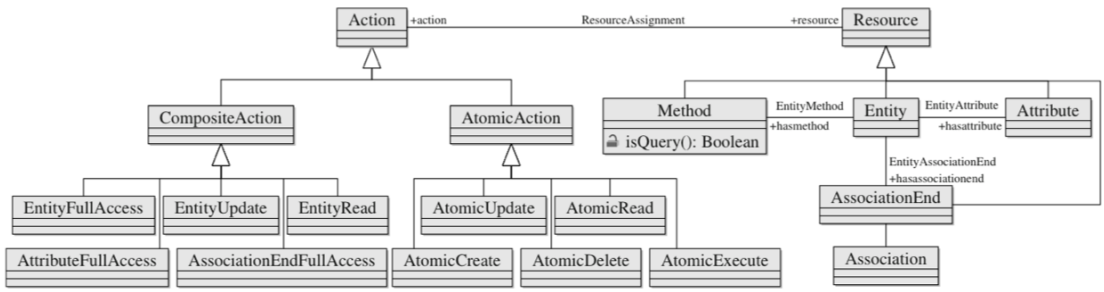

Figure 5-4

Los tipos del modelo que representan a los recursos protegidos serán, en este caso *Entity* así como sus *Attribute, Method* y *AssociationEnd.*

Las acciones que se pueden realizar quedan jerarquizadas según el orden establecido en la figura.

Las acciones atómicas son las que se van a corresponder con las operaciones del sistema real. Las acciones compuestas servirán para agrupar una o más acciones primitivas y darles así una relación de jerarquía.

La política por defecto para las acciones que no poseen permisos explícitos ha de ser definida. Teniendo en cuenta que corremos el riesgo de no permitir la ejecución de métodos seguros en caso de poner la opción por defecto a "denegar" y en el caso contrario corremos el riesgo de que el olvido de asignarle un permiso a una operación permita a éste ejecutar su código sin tener en cuenta la política de seguridad. Hemos optado por continuar las generalidades descritas por Manuel Clavel, es decir, aplicar permisos y Roles por defecto para garantizar la seguridad en todo momento.

5.2 Metamodelo Basic Aspect

Buscamos un subconjunto común a los lenguajes de programación con aspectos, lo que nos permitirá establecer una transformación que sea genérica de tal forma que podamos cambiar la plataforma específica del aspecto generado. Utilizaremos únicamente un subconjunto muy reducido, aquel necesario e imprescindible para poder garantizar toda la funcionalidad requerida en la implementación de la política de seguridad.

Los elementos del metamodelo son: *ResClass* (representando un recurso), *ResAttribute* (representando un atributo del recurso), *ResMethod* (que representa un método), *ResGetMethod*, *ResSetMethod*, *Aspect*, *Pointcut*, *BeforeAdvice, Role-Class* y *Env*. Las metaclases *ResClass, ResAttribute* y *ResMethod* representan elementos de la clase abstracta generada después de la aplicación de la función de transformación a la política de SecureUML. Un *ResMethod* tiene dos subclases: *ResGetMethod* y *ResSetMethod*. Las metaclases *Aspect*, *Pointcut* y *BeforeAdvice* representan elementos del aspecto generado. La metaclase *RoleClass* representa Roles en forma de clases. La metaclase *Env* representa el entorno que proporciona información en tiempo de ejecución como el rol y el nombre del usuario actual. Una instancia de *ResClass* puede tener varios *ResAttributes* y *ResMethods*. Una instancia de un *Aspect* puede tener varios *Pointcuts* y cada *Pointcut* tiene un único *BeforeAdvice*. De tal forma que una instancia de Aspect debe estar relacionada con un único ResClass y cada *Pointcut* debe estar asociado a un único *ResMethod*. La metaclase *RoleClass* se relaciona consigo misma.

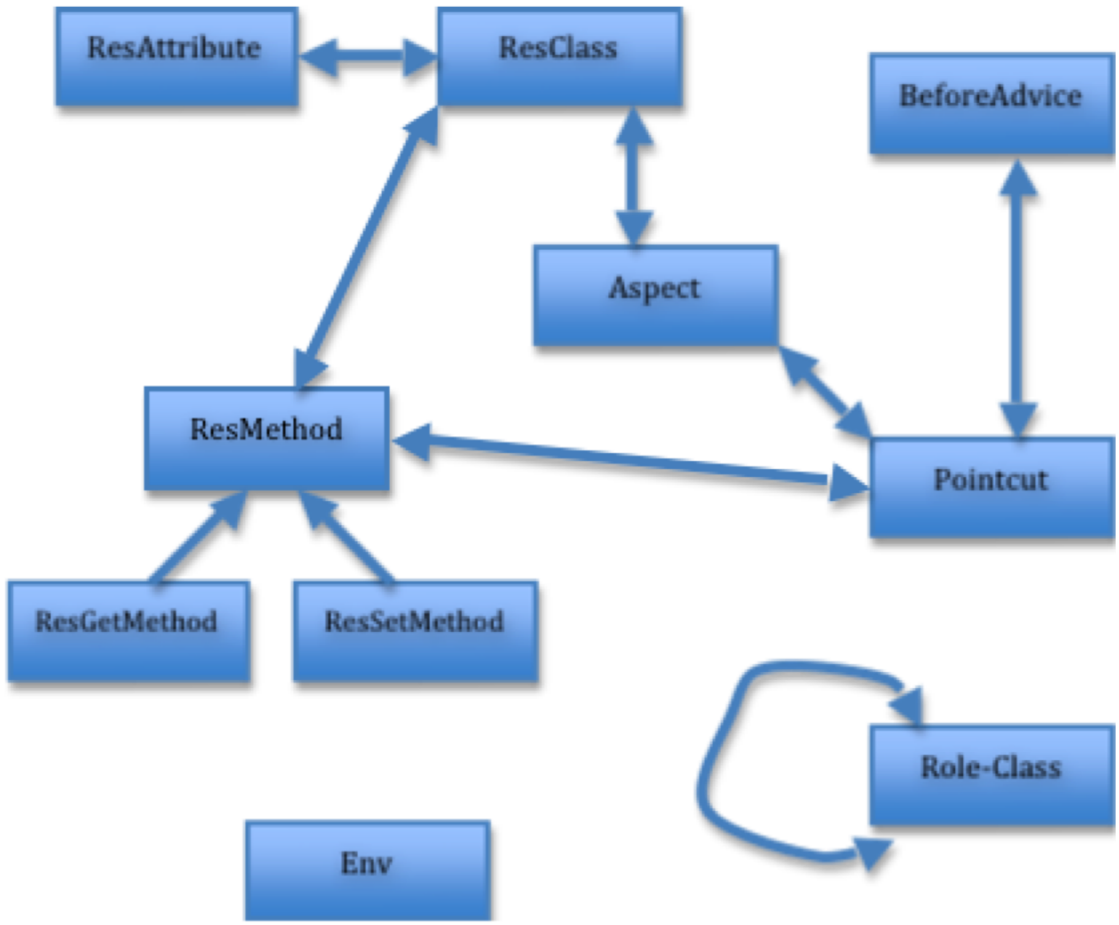

Figure 5-5

5.3 Metamodelo Combinado

Las clases en el metamodelo combinado de SecureUML y Basic Aspect viene dada por la unión disjunta de las clases en ambos metamodelos. Las relaciones entre las clases de ambos metamodelos serán las relaciones ya existentes añadiendo posteriormente las relaciones que asocien las clases de cada metamodelo, especificando de esta forma las propiedades que deben cumplir para que una instancia del metamodelo combinado esté bien formado. Esto es, si un modelo concreto de Basic Aspect es una clase abstracta y un aspecto para la política expresada en el modelo de SecureUML, en términos de los invariantes definidos en el metamodelo combinado.

En la figura Figure 5-6 podemos observar como el Aspecto se asocia con una única instancia de *ResClass* y una *Entity*. Un *ResMethod* se asocia con únicamente un *Resource*, y un *ResAttribute* se asocia con un único *Attribute*. *ResMethod* se asocia también con exclusivamente un *Pointcut.*

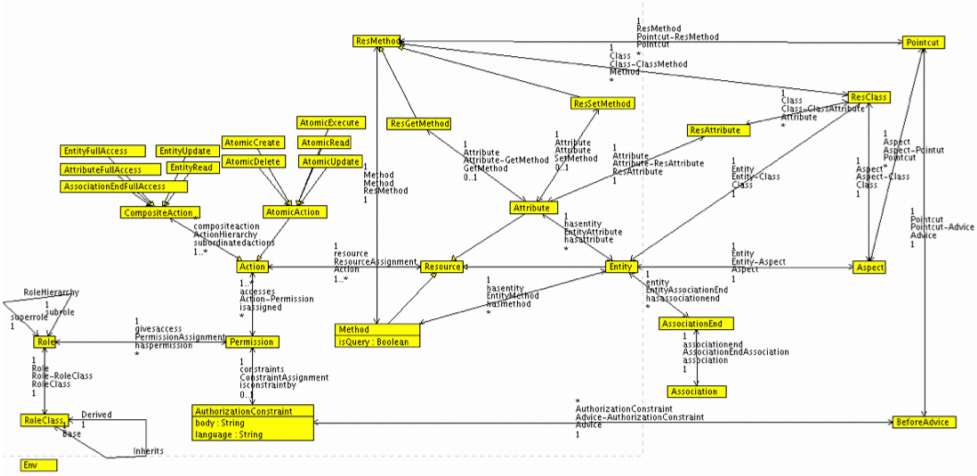

Figure 5-6

6 Algoritmo de trasformación

En este momento vamos a centrarnos en la explicación paso a paso que debe realizar el algoritmo para poder conseguir, a partir de los posibles diferentes modelos de objetos en SecureUML de partida, la generación correcta del aspecto que controla el Access Control para cada modelo de partida.

El algoritmo en un primer paso consigue generar la unión de los modelos de SecureUML y Basic Aspect. Una vez obtenida la unión, en el segundo paso podemos transformar la sintaxis abstracta de Basic Aspect. En nuestro caso al utilizar AspectJ, en este paso final transformaremos la sintaxis abstracta de Basic Aspect en la sintaxis concreta de AspectJ.

Hay que tener en cuenta que también produce de salida 2 archivos más para cada Entity del modelo SecureUML de partida:

> El primero representa la interfaz de environment. En la aplicación final, sobre la que sea utilizado el aspecto de Access Control, tendrá que ser implementada dicha interfaz. Se debe principalmente a que en tiempo de ejecución el aspecto se genera dinámicamente y su constructor no puede llevar parámetros. Entonces de ninguna manera el aspecto puede tener una referencia global al rol del usuario. Por ello la interfaz environment tendrá un método estático que devolverá el rol del usuario actual y cuando el aspecto necesite saber el rol del usuario actual, consultará dicho método para obtenerlo.
>
> El segundo y no menos importante es la clase abstracta que representa la Entity de SecureUML. El Aspecto de Access Control generado capturará jointpoints sobre dicha clase abstracta. En las aplicaciones finales donde sea utilizado el aspecto, dicha clase abstracta tendrá que ser heredada por otra que implementará toda la funcionalidad. A través de enlace dinámico serán capturados los jointpoints igualmente.

Cuando se estudie la implementación y la arquitectura entraremos en detalles de más bajo nivel de abstracción del algoritmo de transformación.

Presentamos el algoritmo de forma general:

Entrada:

Fichero de texto: contiene el/los posibles/s diagrama/s de objeto/s de SecureUML de partida del algoritmo.

Salida:

Fichero Env que representa la interfaz para obtener el rol del usuario actual.

Por cada diagrama de objetos que exista en el fichero de entrada:

- Un fichero con el código generado del aspecto de Access Control con la sintaxis concreta de AspectJ.

- Un fichero con el código generado de la clase abstracta que representa la Entity de SecureUML.

Algoritmo:

Comprobación inicial como precondición del algoritmo. Todos los invariantes descritos sobre el metamodelo de SecureUML se tienen que cumplir sobre el/los diagrama/s de objetos de entrada de SecureUML. Los invariantes se pueden encontrad definidos en Apéndice III: Interfaces de los componentes de la implementación del transformador. Si la corrección no se cumple el algoritmo serializa un error y no empieza ya que el modelo de entrada es incorrecto, si se cumple empieza el algoritmo.

Primer paso. Trasformación de la sintaxis abstracta de SecureUML a la sintaxis abstracta de Basic Aspect, a la vez que realizamos la trasformación, unimos los dos modelos creando el merged:

1. Por cada Entity generamos su Resource Class y se asocian.

2. Por cada Attribute generamos un protected Resource Attribute en la Resource Class asociada con la entidad del atributo. Se asocia con la Resource Class y con Resource Attribute.

3. Por cada Attribute generamos un public Resource Get Method en la Resource Class asociada con la entidad del atributo. Se asocia con Resource Class y con Resource Get Method.

4. Por cada Attribute generamos un public Resource Set Method en la Resource Class asociada con la entidad del atributo. Se asocia con Resource Class y con Resource Set Method.

5. Por cada método generamos un public Resource Method en la Resource Class asociada con la entidad del método. Se asocia con Resource Class y con Resource Method.

6. Por cada Resource Class se genera un Aspect y se asocian.

7. Por cada Aspect se generan sus Pointcut:

    7.1. Por cada método generamos un Pointcut en el aspecto asociado con el Resource Method de la Resource Class y se asocian.

    7.2. Por cada Pointcut generamos un BeforeAdvice sobre el mismo pointcut y se asocian.

7.3. Por cada BeforeAdvice realizamos la asociación con los authorization constraint asociados con el Resource asociado con el Resource Method asociado al Pointcut, distinguimos casos:

7.3.1. Si el Resource es un Attribute y el Resource Method es un Resource Get Method entonces los authorization contraints tienen que estar relacionados con read permissions, en los que se engloban permisión para acciones de AtomicRead, Attribute Full Access, Entity Read y Entity Full Access.

7.3.2. Si el Resource es un Attribute y el Resource Method es un Resource Set Method entonces los authorization contraints tienen que estar relacionados con write permissions, en los que se engloban permissions para acciones de AtomicUpdate, Attribute Full Access, Entity Update y Entity Full Access.

7.3.3. Si el Resource es un Method entonces los authorization contraints tienen que estar relacionados con permissions para acciones de AtomicExecute, AtomicCreate y AtomicDelete.

7.3.4. Es muy importante que cuando se asocien los authorization constraint con el Before Advice, estos deben ser ordenados de acuerdo con los privilegios que tienen los roles a los que se llega navegando a través de las asociaciones (el authorization constrain esta asociado a un permiso y este permiso a un Role). Un Role A tiene más privilegios que un Role B si A está en la relación subrole de B. A partir de esta ordenación se generará el cuerpo del Before Advice.

Como aclaración al último punto del algoritmo (7.3.4). Esta proposición es muy importante en la generación de sintaxis concreta del cuerpo del before advice del pointcut. Tras la ordenación de los authorization constraints, según la prioridad del Role, se hacen comprobaciones anidadas:

Primero es que el usuario posea el Role que está asociado al authorization constraint y las siguientes comprobaciones, anidadas a esta primera, son las condiciones de los authorization constraints asociados al Role. Condiciones vacías serán interpretadas con *true*.

Si se cumple la condición, con la cláusula *return* saldremos del aspecto.

Si no se cumple la condición de los authorization constraint se serializa una excepción que indique que no se ha cumplido el permiso.

Al final del cuerpo comprobaciones anidadas se concatena (por el final) una serialización que indique que el usuario no tiene un rol apropiado para ejecutar el método concreto.

Antes de comenzar el segundo paso, se comprueba la corrección del modelo merged a través de los invariantes definidos en A través de la API de OCLEval, se puede crear un proceso Maude sobre el que:. Si la corrección no se cumple el algoritmo se aborta, si se cumple se continua.

Segundo paso. Transformación la sintaxis abstracta de Basic Aspect a la sintaxis concreta de AspectJ.

8. Por cada Resource Class generamos una declaración de clase abstracta: *public abstract class <nombre>.* Obtenemos <nombre> desde Entity del modelo de partida.

   8.1. Por cada Resource Attribute asociado a la Resource Class generamos su declaración *protected <tipo> <nombre>;.* Obtenemos <tipo> y <nombre> desde Attribute del modelo de partida.

   8.2. Por cada Resource Get Method asociado a la Resource Class generamos una declaración de método*: <tipo> get<nombre>() { return <nombre>;}* . <tipo> y <nombre> se obtienen desde Attribute del modelo de partida.

   8.3. Por cada Resource Set Method asociado a la Resource Class generamos una declaracion de método: *void set<nombre>(<tipo> <nombre>) {this.<nombre> = <nombre>;}* . <tipo> y <nombre> se obtienen desde Attribute del modelo de partida.

   8.4. Por cada Resource Method asociado a la Resource Class generamos una declaracion de método: *public abstract <tipo> <nombre>(<parameters>) {this.<nombre> = <nombre>;}*. <tipo> <nombre> y <parameters> se obtienen desde Method del modelo de partida.

Generamos un fichero de texto por cada Resource Class que contendrá todas estas últimas declaraciones generadas y que representará la Entity transformada a clase abstracta.

Generamos el fichero Env.

9. Por cada Aspect generamos la declaracion de AspectJ: *public aspect <nombre>_ACAspect*. Obtenemos <nombre> desde Entity del modelo de partida.

9.1. Por cada Pointcut generamos la declaración propia de pointcut de AspectJ. La sintaxis de un pointcut en AspectJ se explicó en apartados anteriores. La única consideración importante es que el nombre del pointcut se le pondrá el prefijo de get si el Resource Method es Resource Get Method o set Resource Set Method. Los nombres, tipos, parámetros… serán obtenidos desde Method o Attribute (según corresponda) desde el modelo de partida.

9.2. Por cada Before Advice, generamos la declaración propia parametrizada por la declaración del Pointcut con el que esta asociado. La sintaxis de un befote advine en AspectJ se explicó en apartados anteriores. El cuerpo de la declaración será en que se generó en el paso 7.3.4.

Generamos un fichero de texto por cada Aspect que contendrá el aspecto de control de acceso con la sintaxis concreta de AspectJ. Se vuelve a recalcar la importancia de la interfaz definida en el archivo Env, ya que el aspecto generado continuamente realiza llamadas a dicha interfaz para obtener el Role del usuario actual.

6.1 Arquitectura de la implementación

En este apartado entramos a fondo en la explicación de la arquitectura de componentes utilizada para la elaboración del prototipo de transformación. Iremos presentando componente a componente, explicaremos detenidamente la funcionalidad de cada uno y como se apoyan unos con otros. Por último explicaremos sus implementaciones internas.

Cada componente de nuestro prototipo implementa toda su funcionalidad en un archivo .java que puede tener una o varias clases y obtiene sus servicios importando paquetes de clases tanto de la API de Java y de otros paquetes incluidos en el prototipo.

En el nivel más bajo tenemos la componente OCLEval. Dicho componente tiene una API para conectar Java con ITP/OCL, en el apartado de implementación será explicado con mayor nivel de detalle. Presentamos un esquema de este nivel en la Figure 6-1 :

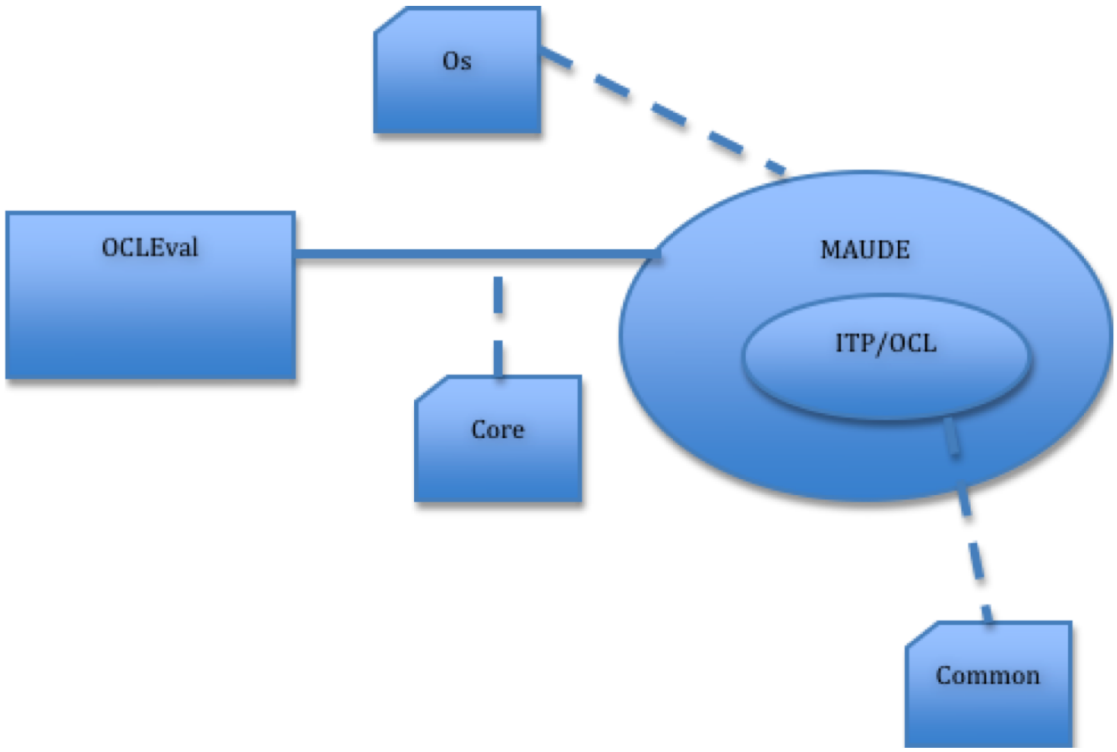

Figure 6-1

La API de OCLEval es capaz de crear un proceso Maude desde Java y poderle mandar peticiones y recibir respuestas de ese proceso. Para todo el proceso de comunicación, OCLEval se apoya en las funcionalidades que brinda el paquete Core. Es importante recalcar la creación del proceso Maude. En el paquete Os tenemos tres versiones de Maude: versión Windows, versión UNIX y versión MAC. Cuando se genere un objeto OCLEval este mandará mensajes a Core para que cree el proceso. Desde Core se consultará el sistema operativo que se está utilizando, con lo cual podrá saber que versión de Maude contenida en Os se debe arrancar en el nuevo proceso a crear. Una vez creado el proceso Maude, Core manda cargar ITP/OCL desde Comon, ya que su fichero .maude se almacena allí. Un gran hándicap al haber utilizado Java como lenguaje y en el paquete Os tener versiones de Maude para Windows, UNIX y MAC es que explotamos al máximo la característica de multiplataforma de Java en nuestro prototipo de transformador.

Como vemos el paquete Os y Comon solo participan en el momento de creación de un objeto OCLEval, en cambio el paquete Core se utilizara continuamente durante la vida de dicho objeto, ya que a través de él se

realizan todas las peticiones, consultas y se obtienen las respuestas del proceso Maude creado.

El siguiente nivel viene dado por el componente SUMLEval. Se indica en la Figure 6-2 . Dicho componente esta asociado con el componente OCLEval. Siempre que se cree un objeto de SUMLEval se creará otro de OCLEval para que el objeto SUMLEval pueda conectarse con ITP/OCL.

El objeto SUMLEval, tras la creación correcta del objeto asociado OCLEval, manda a ITP/OCL el metamodelo completo de SecureUML. Además proporciona una API para trabajara nivel de modelo SecureUML: crear diagramas de objetos, crear invariantes, realizar querys OCL el diagrama de objetos…Más adelante explicaremos la implementación interna completa, explicando la clase SUMLEval extensivamente.

En el nivel más alto se encuentra el componente SecUMLToACAspect. Se puede ver en la también en la Figure 6-2:

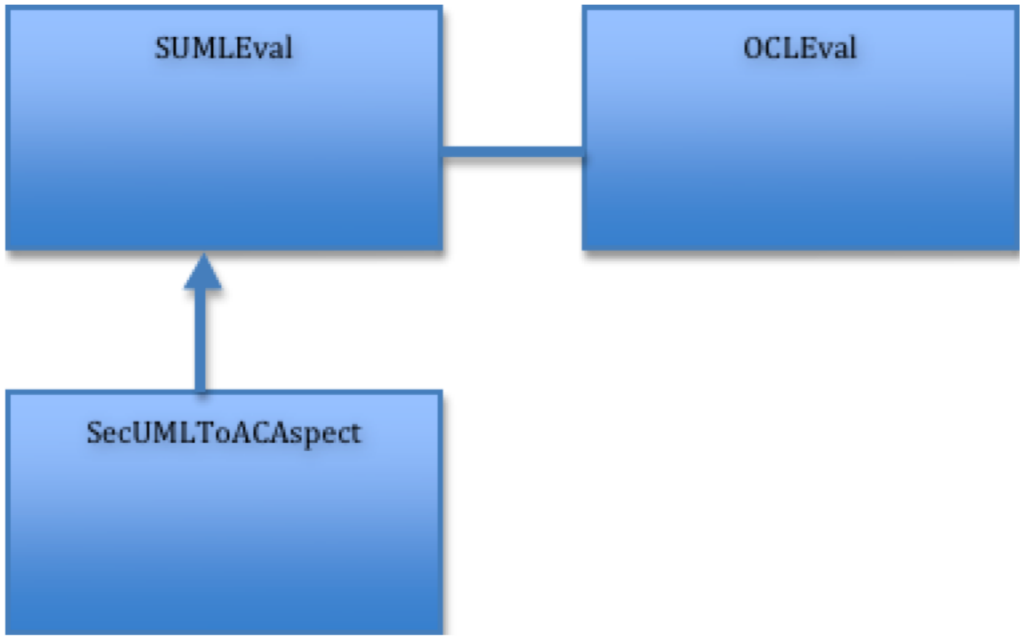

Figure 6-2

Dicho componente posee la clase principal SecUMLToACAspect que generaliza el comportamiento de SUMLEval. Esta clase generaliza el comportamiento de crear el metamodelo de SecureUML. Se encarga también de generar el diagrama de objetos de SecureUML, es decir, el punto de partida para el algoritmo de transformación. Además posee la colección de clases necesarias para crear el modelo merged (en memoria) para realizar la transformación del diagrama de partida al modelo merged (SecureUML + Basic Aspect).Todas esas clases están asociadas a SecUMLToACAspect, ya

que todas las preguntas que deben realizar al modelo de objetos se realizan con la ayuda de la API de SUMLEval (SecUMLToACAspect hereda dicho API). Cuando este formado dicho modelo y comprobado, se mandará imprimir los archivos de salida, tal y como se explica en el algoritmo.

Para finalizar la presentación de la arquitectura y antes de ver la implementación de cada componente detalladamente ofrecemos la visión general de la arquitectura en la Figure 6-3

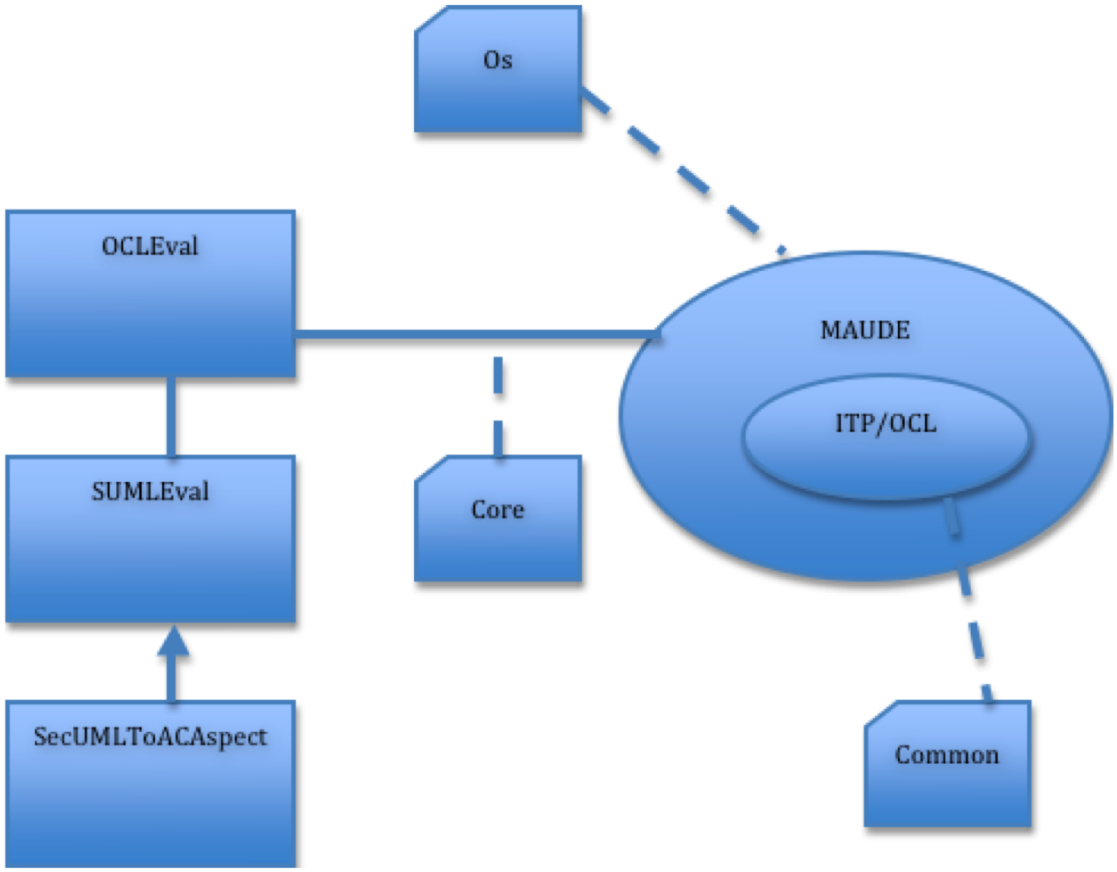

Figure 6-3

6.2 Implementación

A lo largo de este apartado vamos a explicar detenidamente toda la implementación del algoritmo de transformación llevada a cabo dentro del prototipo.

El componte OCLEval. Es un evaluador OCL y está implementado en la clase OCLEval. Dicho componente encapsula, con tecnología Java y a través de una API bien definida, ITP/OCL.

A continuación vamos a proceder a explicar qué es ITP/OCL.

ITP/OCL es una herramienta experimental desarrollada en Maude y creada por Manuel Clavel y Marina Egea. Ha sido desarrollada dentro de nuestra propia facultad, la faculta de informática de la Universidad Complutenses de

Madrid. En su desarrollo se ha beneficiado de las capacidades reflexivas que ofrece Maude. ITP/OCL es una herramienta "rewriting-based tool". Soporta la inspección, el análisis y validación automática de diagramas de clases UML con OCL constraints. Lo más destacado que puede hacer ITP/OCL es:

- Crear diagramas de clases (metamodelos) y diagramas de objetos.
- Introducir operaciones OCL sobre los metamodelos.
- Introducir invariantes OCL sobre en los metamodelos.
- Comprobar los invariantes OCL sobre el diagrama de objetos.
- Realizar preguntas OCL sobre el diagrama de objetos.

Omitimos una explicación más extensa. Una referencia básica de cómo utilizar la herramienta la encontramos en:

(Clavel & Egea)

A través de la API de OCLEval implementado:

- creamos un proceso Maude.
- cargamos ITP/OCL.
- creamos el metamodelo de SecureUML.
- Cargamos el diagrama de objetos de SecureUML de entrada de la transformación.
- Realizamos preguntas OCL al diagrama de objetos existente en ITP/OCL.

Resaltamos de la API la claridad con la que obtenemos resultados, es decir, por ejemplo a realizar preguntas la API no nos devuelve toda la salida que produce Maude (producía la pregunta realizada y la respuesta), sino que solo nos retorna una String con la respuesta bien parseada a través de métodos privados que borran todo lo innecesario de la salida estándar de Maude. Un método public importante es:

```
public List<String> colToList(String answer)
```

Cuando realizamos una pregunta que de primeras sabemos que va a devolver una colección, la respuesta obtenida por el método *query* de la API es una String que representa como se construye una colección en Maude. La construcción se origina con la constructora *col*. Si le pasamos dicha respuesta como parámetro a *colToList,* internamente este método transforma el parámetro de entrada en una List<String> omitiendo *col*. La API completa y detallada se encuentra en Apéndice III: Interfaces de los componentes de la implementación del transformador..

Una vez explicada la API que nos brinda OCLEval se va a explicar como a través de invariantes podemos comprobar que el algoritmo de transformación, explicado en el punto anterior, puede ser validado. Se mostrarán invariantes en notación OCL y el mapeo que realiza ITP/OCL, para ver y entender ambas formas de representar los invariantes.

Sabemos que al principio de la transformación partimos de un archivo de entrada con el diagrama de objetos de una instancia particular de SecureUML. Antes de comenzar la transformación, el metamodelo de SecureUML será cargado en ITP/OCL utilizando el componente OCLEval. Posteriormente, también con el API ofrecida por OCLEval, cargaremos en ITP/OCL el diagrama de objetos representado en el archivo de entrada. La implementación del prototipo genera errores si no hay concordancia entre el metamodelo de SecureUML y el diagrama de objetos de entrada. Aun así, podríamos definir invariantes sobre el metamodelo de SecureUML. Serían incorporados a ITP/OCL a la vez que se introduce el metamodelo de SecureUML y una vez introducido el diagrama de objetos se comprobarían. Si la comprobación es correcta seguiríamos con la transformación ya que hemos validado el diagrama de objetos de partida gracias a los invariantes. Si la comprobación es incorrecta no realizaríamos la transformación porque sabemos a ciencia cierta que el diagrama de objetos de partida es incorrecto.

En (Doser, Clavel, Basin, & Egea, 2008) se encuentran los invariantes sobre SecureUML. Mostramos un par de ejemplos de invariantes sobre SecureUML, con notación TP/OCL, que indican que tienen que existir un Role y un Permiso por defecto en el modelo de objetos de SecureUML:

```
(insert-invariant SUMLMETAMODEL::

  ((((Role #allInstances) ->select ( (r:Role )| ((r:Role) #default ) )) ->size ( )) = (1 )) .)

(insert-invariant SUMLMETAMODEL::

  ((((Permission #allInstances) ->select ( (p:Permission )| ((p:Permission) #default ) )) ->size ( )) = (1 )) .)
```

Con los métodos para introducir invariantes de la API que ofrece OCleval (contextuales o no), podríamos incorporarlos en la implementación. Ver Apéndice III: Interfaces de los componentes de la implementación del transformador.

Nuestra implementación hace que el modelo de Basic Aspect se genere en memoria a través de clases Java. Cada clase de modelo merged no correspondiente a clases de SecureUML será implementada con una clase Java. Si quisiésemos haber validado el modelo merged más formalmente, a través de invariantes, tendría que haber existido una copia de dicho modelo que se crea en memoria (las clases JAVA descritas anteriormente) también en ITP/OCL. Gracias a ello podríamos haber incorporado y validado los invariantes para el modelo merged. Si la validación hubiese sido correcta procederíamos a imprimir los archivos de texto con el código generado y si no serializaríamos un error y detendríamos la transformación. Mostramos un par de invariantes OCL de ejemplo:

Sobre Entity. Sabemos que cada Entity se asocia con ResClass:

```
Context Entity inv:
Entity.allInstances->forAll(e | e.class = e.ResClass.class)
```

Sobre Attribute. Solo puede estar asociado con una Entity y no con varias:

```
Context Attribute inv:
Attribute.allInstances->forAll(a | a.entity.class.attribute->select(ca | ca.name = a.name)->size() =1))
```

El componente SUMLEval. La API de dicho componente está implementada en la clase SUMEval. Lo primero reseñable sobre la implementación en esa clase es que todos los elementos para crear el metamodelo de SecureUML son contantes String. Veamos un par de constantes:

```
protected static final String ATTRIBUTE = "Attribute" ;

protected static final String METHOD = "Method" ;
```

En el propio constructor de la clase creamos el objeto OCLEval y mandamos a ITP/OCL el metamodelo de SecureUML. Cada llamada a la API OCLEval nos devuelve un resultado en String que iremos mostrando por la salida estándar para comprobar que todo el proceso de creación del metamodelo de SecureUML se está realizando correctamente. En las ultimas líneas se introducen operadores que serán muy útiles para querys que tengan que ver con transitividad, por ejemplo ver que un Role tiene más privilegios que otro Role.

El siguiente paso, una vez creado el metamodelo, es crear el diagrama de objetos siguiendo la API descrita en Apéndice III: Interfaces de los componentes de la implementación del transformador..

El primer método que hay que ejecutar de la API es: `public String createSecurityDiagram(String diagram) throws Exception .` Dicho método envía el mensaje a OCLEval para que construya el diagrama de objetos y inserta ya en él todos los objetos que tiene que haber por defecto, tal y como deben existir en SecureUML. Posteriormente se utiliza la API para insertar los objetos en el diagrama creado. Recalcamos que la API ofrece diferentes tipos de preguntas OCL al diagrama. Toda la creación del modelo merged se basa en preguntas OCL al diagrama. Además, para crear los archivos de salida del algoritmo, continuamente se utilizará la API de querys para que devuelva la información que queremos obtener del diagrama de objetos SecureUML representado en ITP/OCL.

Para finalizar con la implementación presentamos la implementación del componente principal.

El componente SecUMLToACAspect. Se encarga de implementar a través de un conjunto de clases el modelo merged. Es conveniente leer la sección donde se explica detalladamente el algoritmo de transformación. La explicación que vamos a tratar sigue la traza de ejecución de la transformación, que sigue los pasos explicados en el algoritmo de transformación.

La primera clase es *SecUMLToACAspect*. Dicha clase es una extensión de la clase *SUMLEval* que representa el componente con el mismo nombre. La funcionalidad implementada la clase SecUMLToACAspect se divide en 3 rasgos:

La primera es la funcionalidad heredada de SUMLEval, es decir, la creación del metamodelo SecureUML en ITP/OCL y toda la API que

ofrece SUMLEval para crear diagramas de objetos y realizar querys a dicho diagrama.

La segunda es que dicha clase en su constructor manda leer desde fichero de texto el diagrama de objetos y a través de la API heredada crea el diagrama de objetos que será el modelo de partida del algoritmo de transformación. Un ejemplo de fichero de texto se encuentra en el Apéndice I : Modelo completo de la política de seguridad aplicada a TRC. Entrada al Transformador. Vamos a explicar como debe de crearse dicho archivo de texto:

- La primera regla es que habrá una instrucción por cada línea.
- La segunda es que la primera línea será:

```
createSecurityDiagram("Nombre del diagrama");
```

- En las posteriores líneas podremos poner para la creación del diagrama de objetos:
  - `insertRole("Nombre del Role") ;`
  - `insertRoleHierarchy( "Nombre del subRole","Nombre del superRole");`
  - `insertUser("Nombre del User");`
  - `insertUserAssignment(Nombre del User","Nombre del Role");`
  - `insertEntity( "Id de la Entity","Nombre de la Entity") ;`
  - `insertAttribute( "Id de la Entity", "Id del Attribute","Nombre del Attribute","Tipo del attribute") ;`
  - `insertMethod( "Id de la Entity", "Id del Method","Nombre del Method","Tipo del Method","Parametros del Method", true o false según sea o no query el Method) ;`
  - `insertAuthConstraint( "Nombre del AuthConstraint", mkValue("Java"), mkValue("cuerpo de la condición en lenguaje Java, si es vacía se pone true")) ;`
  - `insertPermission( "Nombre del permiso") ;`
  - `insertPermissionAssignment( "Nombre del permiso", "Nombre del Role") ;`

- `insertConstraintAssignment("Nombre del AuthConstraint","Nombre del permiso");`
- `grantEntityFullAccess( "Id de la Entity", "Nombre del permiso");`
- `grantEntityReadAccess( "Id de la Entity", "Nombre del permiso");`
- `grantEntityUpdateAccess( "Id de la Entity", "Nombre del permiso");`
- `grantAtomicCreateAccess( "Id de la Entity", "Nombre del permiso");`
- `grantAtomicDeleteAccess( "Id de la Entity", "Nombre del permiso");`
- `grantAttributeFullAccess "Id del Attribute", "Nombre del permiso");`
- `grantAtomicUpdateAccess( "Id del Attribute", "Nombre del permiso");`
- `grantAtomicReadAccess( "Id del Atrribute", "Nombre del permiso");`
- `grantAtomicExecuteAccess( "Id del Method", "Nombre del permiso");`

La tercera funcionalidad es, que una vez cargado el diagrama de objeto desde el fichero a ITP/OCL y comprobado que es correcto, realiza una query para ver los objetos Entity que existen en el diagrama de objetos. A partir de la lista que obtiene empieza a ejecutar el algoritmo de transformación para crear el modelo merged, comprobarlo y crear los ficheros de salida por cada Entity.

La siguiente clase a explicar es la que mapea Resource Class. La clase se llama *RClass*. *SecUMLToACAspect*, como se ha citado arriba, manda crear un objeto de dicha clase por cada Entity presente en el diagrama de objetos para comenzar a crear el modelo merged en memoria. En el constructor de esta clase se realizan querys para obtener los atributos y métodos de la Entity (únicamente sus identificadores) y se crean pertinentemente los objetos de *RAttribute*, *RMethod*, *RGetMethod* y *RSetMethod.* Una vez mapeada en memoria toda la Entity (creados todos los Resources) mandamos construir el objeto ACAspect.

RAttribute mapea Resource Attribute.

RMethod mapea Resource Method.

RGetMethod mapea Resource Get Method. Hereda de la clase *RMethod*.

RSetMethod mapea Resource Set Method. Hereda de la clase *RMethod*.

Cuando se manda construir el objeto ACAspect se observa claramente que ya el puente entre los modelos de SecureUML y Basic Aspect se ha realizado gracias a la creación de los objetos Resource. Continuamos describiendo *ACAspect* que reprenda Aspect del modelo Basic Aspect. En el proceso de construcción del objeto se obtienen todos los *RMethod* (por polimorfismo también se encuentran los *RGetMethod* y *RSetMethod*) y para cada uno se manda crear un objeto *ACPointCut*.

ACPointCut representa a Pointcut del modelo Basic Aspect. Como se explicó en el algoritmo, el Pointcut tiene que ver modificado su nombre añadiendo el prefijo *set* o *get* si corresponde a getter o setter. Dado la existencia de polimorfismo en *RMethod*, a través de *instanceof* concluiremos si ese método era un getter o setter y ya en la propia construcción del objeto *ACPointCut* resolveremos dicha cuestión. Antes de finalizar la construcción mandamos crear el objeto *ACBeforeAdvice* que representa el before advice de dicho Pointcut.

ACBeforeAdvice representa a BeforeAdvice del modelo Basic Aspect. Lo más importante es que en la creación del objeto vamos a crear correctamente el cuerpo del befote advice. Como se proponía en el algoritmo, tendremos que distinguir según el tipo de método (utilizando *instanceof*), los authorization constraints a obtener a través de una query al diagrama de objetos:

RGetMethod -> Read Authorization Constraints.

RSetMethod -> Write Authorization Constraints.

RMethod -> Method Authorization Constraints.

Una vez obtenida la lista de authorization constraints, siguiendo lo que dictaba el algoritmo, los ordenamos con el método de la burbuja. El orden es de mayor privilegios de Role, asociado al authorization constraint, a menor privilegios. Un Role A tiene más privilegios que B si A está en la relación subrole de B. Utilizamos como función para comparar otra query de la API SUMLEval. Esta indica si A tiene más privilegios que B devolviendo un resultado booleano.

Cuando esta ordenada la lista se manda crear el cuerpo del before advice. Si solo existe el authorization constraint por defecto, se activa un flag que indicará a la hora de imprimir no lo haga, ya que dicho método no tiene ninguna restricción luego no es necesario que el aspecto de Access Control tenga un pointcut y un before advice asociado que no hagan absolutamente

nada. Si existiesen diversos authorization constraints la forma de crear el cuerpo es la siguiente (partimos de que ya están ordenados):

Se crea una condición para comprobar el Role y dentro se anidan las comprobaciones de los cuerpos de los authorization constraints asociados a dicho Role de la lista de authorization constraint.

Las comprobaciones de los authorization constraint hacen que si se cumplen se ejecute *return* si no se serializa una excepción que indique que ha habido un error de permisos.

Al final se serializa una excepción de Role incorrecto.

Veamos un ejemplo. Se ha extraído del Apéndice II: Salida del Transformador para la política de seguridad aplicada a TRC. . Es el before advice creado para el método de crear un Trc. Al principio se obtiene el Role del usuario actual. En el bucle siguiente se guardan los atributos de llamada del método para que puedan ser mostrados en la generación del error. Luego se observan las comprobaciones de Role y cuerpos de los authorization constrains con if anidados y las serializaciones de excepciones, tal y como se ha explicado anteriormente:

```
//obtener el Role actual

caller=Env.getUsrRole();

//Obtener parametros para notificar el error

String argumentsInError="";

for (int i=0;i<thisJoinPoint.getArgs().length;i++){

      if   thisJoinPoint.getArgs()[i]!=null)
      argumentsInError+="\n\t\t"+
      thisJoinPoint.getArgs()[i].toString();

      else argumentsInError+="\n\t\tnull;";

}

if ( caller instanceof TestSupervisor){

      if ( pOwner.equals(caller.name) ) return ;

      else throw new Exception("Permission error: Role:

            "+caller.getClass().getName()+"\n In:
            "+thisJoinPoint+"\n Attributes:
            "+argumentsInError);

}

if ( caller instanceof TestOperator){

      if ( pOwner.equals( caller.name ) && pScope ==

            TypeOfScope.Private ) return ;
```

```
            else throw new Exception("Permission error: Role:

                  "+caller.getClass().getName()+"\n In:

                  "+thisJoinPoint+"\n Attributes:

                  "+argumentsInError);

      }

      throw new Exception("Role error: Role:

            "+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n
            Attributes: "+argumentsInError);
```

En el momento que se hayan creado todo los objetos del modelo merged en memoria, se realizarán comprobaciones que indique que todo el proceso de transformación se ha realizado correctamente. Si es así mandamos a través de *SecureUMLToACAspect* a los objetos *RClass* creados que generen los archivos de salida.

Gracias a las asociaciones del modelo merged entre objetos, se irán llamando a sus métodos *print* según corresponda. En los métodos *print* de cada clase realizamos querys a través de la API heredada. Cada objeto de clases que construyen el modelo merged en memoria únicamente tiene el id que le corresponda del diagrama de objetos. Si queremos saber nombre, tipo, atributos… necesitamos preguntar al diagrama de objetos de SecureUML y obtener los valores correspondientes. Al final, clases resources gracias a los *print* lograran crear el archivo de salida de la clase abstracta sobre la que se aplica el aspecto de Access Control. Clases que se correspondían al modelo de Basic Aspect a través de los *print* generarán el aspecto de Access Control con la sintaxis concreta de AspectJ. No hay que olvidar que también, como explicaba el algoritmo, se generará el archivo de interfaz Env.

7 Aplicación del proceso de transformación a un ejemplo real.

En este capítulo vamos a desarrollar un ejemplo de cierta envergadura a nuestro prototipo de transformador. El ejemplo es de gran relevancia, ya que está extraído de trabajos de una gran empresa del sector tecnológico español. Realizaremos una presentación del modelo SecureUML de partida para TRC. Mostraremos como aplicar nuestra herramienta sobre el modelo completo, presentando entradas y salidas, finalizando realizado pequeños test a la aplicación final.

7.1 Presentación del modelo TRC.

Observemos el diagrama de objetos de SecureUML creado. Le llamaremos diagrama TRC. Se muestra en Figure 7-1:

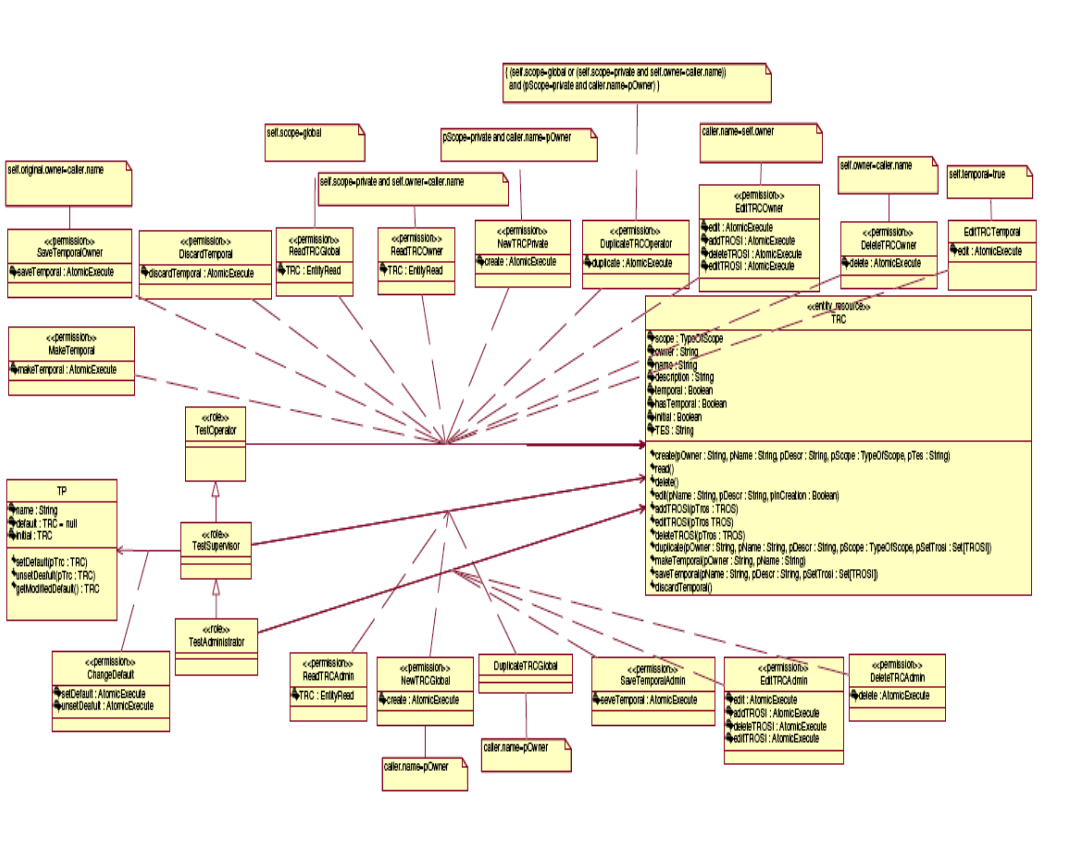

Figure 7-1

Recalcamos que dicho diagrama pertenece a la aplicación desarrollada por la empresa.

En la Figure 7-1 podemos ver el modelo expresado en SecureUML para la política de seguridad de la aplicación. La naturaleza jerárquica del mismo muestra como los roles de menor responsabilidad tienen unas restricciones en muchos casos diferentes a las de aquellos con roles de mayor graduación.

La aplicación desarrollada por la empresa gestionaba el Access Control con una implementación que sabíamos que no utilizaba tecnología Aspect utilizaban la expuesta en (Clavel, da Silva, Braga, & Egea, 2007). En la demostración de nuestra herramienta, vamos a ver que podemos obtener un aspecto de Access Control a partir del diagrama TRC en un primer momento. Con ello demostraremos que la implementación del algoritmo de trasformación obtiene buenos resultados. Para la demostración vamos a ir explicando y mostrando resultados enfocándonos en el método *create.* En un paso siguiente, generaremos una mini-aplicación de testing llamada TRCManager con el aspecto generado para todo el modelo TRC y por diversos ejemplos lograremos ver que el aspecto funciona correctamente.

7.2 Generación de código automática

7.2.1 Generación del Input del prototipo

Para entender como crear el archivo de entrada de la herramienta, recomendamos la lectura del apartado Implementación. A partir del diagrama TRC se creó con las restricciones de nuestro prototipo el archivo de entrada. Se puede consultar en el Apéndice I : Modelo completo de la política de seguridad aplicada a TRC. Entrada al Transformador. . Dicho archivo, en este ejemplo, es el input de nuestro transformador. El archivo tiene como nombre TRC.

Enfocándonos en el método *create* mostramos las líneas del archivo de entrada que lo representa. Para entenderlo más fácilmente consultar la Figure 7-1

Creamos el method. Se observa el id de la entidad con la que se asocia, el id del método, el nombre del método, el tipo de retorno, los parametros y el flag *isQuery*.

```
insertMethod( "Trc", "create","create","void","String pOwner,String
pName,String pDescr,TypeOfScope pScope,String pTes", true) ;
```

Creamos los permisos, dándoles un id:

```
insertPermission( "NewTRCGlobal") ;

insertPermission( "NewTRCPrivate") ;
```

Creamos los authorization constraints de los permisos. Los atributos son: Id, lenguaje del cuerpo del constraint y cuerpo del constraint. Estos dos últimos atributos irán encapsulados en la función *mkValue()*, función de apoyo para que el transformador reconozca bien la entrada.

```
insertAuthConstraint( "NewTRCPrivateAC", mkValue("Java"),
mkValue("pOwner.equals( caller.name ) && pScope == TypeOfScope.Private")) ;

insertAuthConstraint( "NewTRCGlobalAC", mkValue("Java"),
mkValue("pOwner.equals(caller.name)")) ;
```

Se asocian los authorization constraints con los permisos correspondientes:

```
insertConstraintAssignment( "NewTRCGlobalAC", "NewTRCGlobal") ;

insertConstraintAssignment( "NewTRCPrivateAC", "NewTRCPrivate") ;
```

Se asocian los permisos con los roles correspondientes:

```
insertPermissionAssignment( "NewTRCGlobal", "TestSupervisor") ;

insertPermissionAssignment( "NewTRCPrivate", "TestOperator") ;
```

Se asocian los permisos con el método *create*. Al ser un método la acción asociada es AtomicExecute:

```
grantAtomicExecuteAccess( "create", "NewTRCGlobal") ;

grantAtomicExecuteAccess( "create", "NewTRCPrivate") ;
```

7.2.2 Ejecución del prototipo y resultados obtenidos

Ya hemos creado el archivo de partida. Para ejecutar el algoritmo hay que escribir en consola:

> java Main

Una vez ejecutado lanzará un dialogo SWING para abrir un fichero de texto. Seleccionamos el archivo TRC y comienza a ejecutarse el algoritmo de transformación, mostramos toda la salida que produce a lo largo del proceso de ejecución del algoritmo. Se puede observar que la salida se divide en 5 grupos:

- Resultados paso a paso de la creación del metamodelo de SecureUML.
- Resultados paso a paso de la creación del diagrama de objetos TRC.
- Resultados paso a paso del mapeo (creación del modelo merged).
- Resultados paso a paso de la creación de la clase abstracta.
- Resultados paso a paso de la creación del aspecto de Access Control.

Para el ejemplo concreto extraído, todos los resultados para el método *create* se mostraron correctamente.

7.2.3 Output del prototipo

Una vez que todo ha ido correctamente presentamos los tres archivos de Output.

- Trc.java, es la clase abstracta que representa a Trc.
- Env.java, la interfaz para obtener el Role del usuario actual.
- Trc_ACAspect.aj, el aspecto de Access Control.

Los tres ficheros de salida, que produce esta transformación, se pueden consultar en el Apéndice II: Salida del Transformador para la política de seguridad aplicada a TRC.

Continuando con nuestro ejemplo del método *create*.

En Trc.java, al ser *create* un método, se creó el método abstracto correspondiente según la entrada:

```
public abstract void create (String pOwner,String pName,String
pDescr,TypeOfScope pScope,String pTes) throws Exception;
```

La interfaz Env.java siempre es la misma, sea cual sea el modelo de entrada.

En Trc_ACAspect.aj se creo:

El pointcut que captura el jointpoint del método *create* abstracto:

```
pointcut PCcreate( Trc  self , String pOwner,String
pName,String pDescr,TypeOfScope pScope,String pTes): call(void
Trc.create( String , String , String , TypeOfScope , String
))&&target( self )&&args(pOwner, pName, pDescr, pScope, pTes);
```

El before advice asociado al pointcut que representa el Access Control descrito en la entrada del algoritmo. Como se explicó en el apartado Implementación se anidan las comprobaciones de Role y cuerpo de los authorization constraints de mayor a menor privilegio de Role:

```
before (Trc  self , String pOwner,String pName,String pDescr,TypeOfScope pScope,String
pTes) throws Exception:PCcreate( self , pOwner, pName, pDescr, pScope, pTes){

caller=Env.getUsrRole();
String argumentsInError="";

for (int i=0;i<thisJoinPoint.getArgs().length;i++){
                if(thisJoinPoint.getArgs()[i]!=null)
                        argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i].toString()
                ;
                 else argumentsInError+="\n\t\tnull;";
}

if ( caller instanceof TestSupervisor){
        if ( pOwner.equals(caller.name) ) return ;
        else throw new Exception("Permission error: Role:
                "+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
                "+argumentsInError);
        }
        if ( caller instanceof TestOperator){
                if ( pOwner.equals( caller.name ) && pScope == TypeOfScope.Private )
                return ;

                else throw new Exception("Permission error: Role:
                "+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
                "+argumentsInError);
        }
        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
                "+thisJoinPoint+"\n Attributes: "+argumentsInError);
}
```

7.3 Miniaplicación de prueba TRCManager

Durante la fase de desarrollo de cómo generar un aspecto se realizó una pequeña aproximación de aplicación real de la funcionalidad del modelo TRC. Se implemento:

- Una GUI en SWING y AWT.
- Una capa de negocio que contiene una tabla Hash para guardar los TRC creados siendo los nombres de TRC las claves, así no se repiten TRC con el mismo nombre.

- Una capa de persistencia de ficheros de texto para guardar y obtener usuario y Role actual.

- Políticas se seguridad no descritas en el diagrama SecureUML. No puede haber dos Trc con el mismo nombre, todos los User tienen un Trc DEFAULT que nunca puede ser borrado. Se encuentran en (Clavel, da Silva, Braga, & Egea, 2007)

Se compiló todo con javac y se creo un archivo TrcManager.jar. Se ejecuta por consola con:

> java –jar TrcManager.jar

De primeras tenemos que introducir nuestro nombre y Role. Se guardará dichos datos en un fichero. Ver : Figure 7-2

Figure 7-2

Una vez que entremos mostramos la vista general de la aplicación a través de la Figure 7-3 **Error! Reference source not found.**

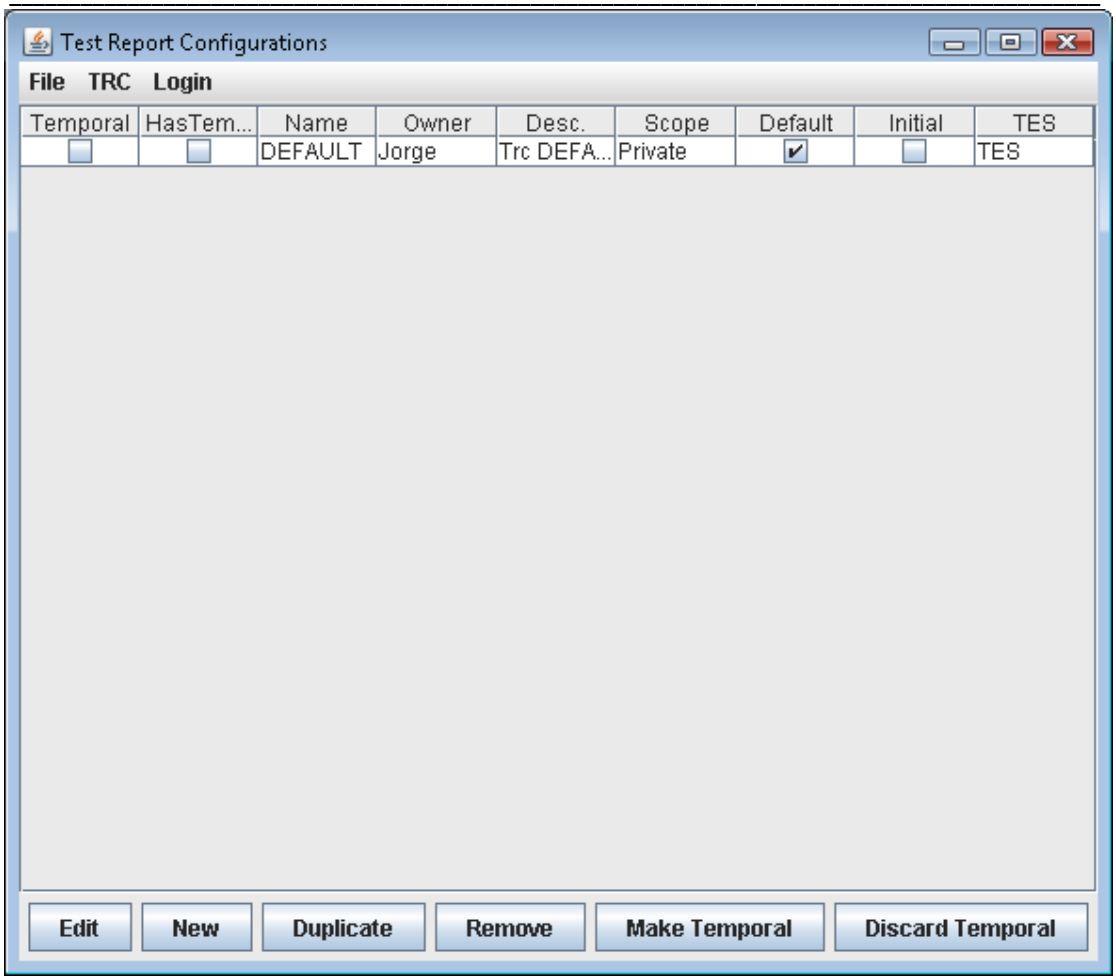

Figure 7-3

Por ahora no existen restricciones de Access Control y cualquier usuario podría hacer todas las funcionalidades del TRC menos los requisitos de seguridad que no provienen de Access Control. Lo que más interesa en estos momentos y detallamos a continuación es cómo en esta aplicación de partida introducimos los archivos generados por nuestro transformador y aplicar el Access Control. Vamos a ver que se realiza en poco pasos, con pocas modificaciones, es decir, de forma muy ortogonal:

- Se introdujo el archivo de interfaz Env. En el método de obtener el Role actual se implementó la lectura de fichero donde se almacenan el usuario y su Role.

- Se introdujo la clase abstracta TRC e hicimos que la que ya existía heredase de ella. Los Jointpoint se capturan en los métodos abstractos de la clase abstracta y se propagan por enlace dinámico a todas las clases que hereden de él.

- Se introdujo el aspecto generado. Como sabemos que el aspecto lanza excepciones el único cambio en el código anterior fue nada más insertar bloques try catch en cualquier llamada a algún método del TRC.

- Volvemos a compilar toda la aplicación con javac y creamos un nuevo TrcManager.jar.

- Para el proceso de entretejido del aspecto por toda la aplicación, compilamos con AspectJ (comando ajc) el archivo TrcManager.jar.

- Uno de los problemas que surge para ejecutar el archivo TrcManager.jar, con el entretejido, es que necesita la maquina del usuario final, además de la JRE, la Aspect Runtime Library (Aspectjrt.jar), situada esta última dentro del CLASSPATH. Para evitar al usuario final tener que obtener dicha librería la hemos incluido en nuestro TrcManager.jar ejecutable. Con ello ya la aplicación encuentra dentro de su .jar la Aspect Runtime Library que necesitaba y no tiene que buscarla por el CLASSPATH.

Finalmente con > java –jar TrcManager.jar ejecutamos ahora de nuevo la miniaplicación pero ahora con Access Control.

Con lo explicado se puede comprobar que con pocos cambios hemos introducido muy fácilmente el Access Control. A un nivel industrial y de una aplicación de envergadura vemos que podría haber un equipo desarrollando la aplicación sin Access Control y otro equipo desarrollando a través de nuestro transformador el aspecto de Access Control. La integración de ambos módulos es muy ortogonal. Una de las grandes características principales de los aspectos es que ofrecen el Access Control de una manera transparente ya que capturan los Jointpoint a través de pointcut y se verifican las restricciones en los before advices. Los desarrolladores de la aplicación sin Access Control implementarán todas las funcionalidades y el aspecto de forma transparente a dicha aplicación primera introducirá las restricciones para ofrecer la aplicación final con Access Control.

A un nivel global, indicando pasos para cualquier aplicación, se asemeja mucho a los pasos que hemos realizado para nuestra aplicación en particular. El único punto conflictivo será el tratamiento de excepciones. En nuestra aplicación hemos hecho que siempre exista un bloque Try Catch y mostremos excepción si se produce. Al mostrar las excepciones las tratamos antes, ya que traen mucha información a nivel de programador, pero trasformamos dicha información a otra más legible y que la entienda mejor el usuario final de la aplicación. Un problema que aparece es que no podemos mostrar Trc de diferentes usuarios que tengan Role de TestOperator ya que el user tiene que ser el owner para poder ver los Trc. Ello genera que cuando se refresca la aplicación errores de que el user no tiene permisos, mostrándose por pantalla. Si se quisiese posibilitar dicha funcionalidad habría que estudiar casos y realizar un mejor tratamiento de excepciones y saber cuando mostrar excepciones o cuando no. Por ejemplo si se produce excepción de que el owner no es el User en vez de mostrarla lo que haríamos es que el Trc no se muestre en la GUI y así dejaría de dar excepciones.

La conclusión más importante. En si aplicar el Access Control generado a cualquier aplicación es muy ortogonal. El único posible problema va a ser el estudio de tratamiento de excepciones por parte de las capas de presentación de las aplicaciones para aplicar las políticas de seguridad que no se reflejan exactamente en el diagrama SecureUML.

Diferentes ejemplos de prueba sobre la miniaplicación para observar que las restricciones de Access Control se cumplen se pueden consultar en el Apéndice IV : Casos de Prueba de TRC Manager

8 Bibliografía

Clavel, M., & Egea, M. (n.d.). *A quick ITP / OCL Tutorial*. From A quick ITP / OCL Tutorial: http://maude.sip.ucm.es/itp/ocl/contents.html

Clavel, M., da Silva, V., Braga, C., & Egea, M. (2007). *Model-Driven Security in Practice: An Industrial Experience.* Lecture Notes in Computer Science.

ACM. (2008). Symposium on Access Control Models and Technologies.

Basin, D., Doser, J., & Lodderstedt, T. (2002). *SecureUML: A UML-Based Modeling Language for Model-Driven Security.* Institute for Computer Science, University of Freiburg, Germany.

Danish Strategic Research Council. (2008). Aspects of Security for Citizens. *Nordic Security Days 2008.* Borgernes.

Doser, J., Clavel, M., Basin, D., & Egea, M. (2008). *Automated Analysis of Security-Design Models.* Information and Software Technology.

Egea, M., & Clavel, M. (2006). *ITP/OCL: A Rewriting-Based Validation Tool for UML+OCL Static Class Diagrams.* From http://maude.sip.ucm.es/~clavel/pubs/clavel-egea06b.pdf

G. Georg, R. F. (2002). An aspect-based approach to modeling security concerns. *Proceedings of the Workshop on Critical Systems Development with UML.* Dresden.

G. Kiczales, J. L.-M. (1997). Aspect-Oriented Programming. *Object-Oriented Programming, 11th European Conference.* Jyvaskyla.

Indrakshi Ray, R. F. (2003). An aspect-based approach to modeling access control concerns. *Information and Software Technology , 46*.

J. Viega, J. B. (2001). Applying Aspect Oriented Programming to Security. *Cutter IT Journal , 14*.

Jacobson, I. (2003). Use Cases and Aspects - Working Seamlessly Together. *Journal of Object Technology , 2*, 28.

Jaeger, T., & Tidswell, J. (2000). *Rebuttal to the NIST rbac model proposal.* ACM. proceedings of 5th ACM Workshop on Role-Based Access Control.

Kulkarni, V., & Reddy, S. (2003). Integrating aspects with Model Driven Software Development. *International Conference on Software Engineering Research and Practice.* SERP.

Kulkarni, V., & Reddy, S. *Separation of concerns in Model Driven Development.* IEEE Software.

Meyer, B. (October de 1992). Applying Design by Contract. *Computer , 25*.

Meyer, B. (1997). *Object-Oriented Software Construction.* Prentice Hall.

OMG. (2003). *MDA Guide Version 1.0.1.*

OMG. (2006). *Object Constraint Language formal/06-05-01.*

(2004). *Role Based Access Control.* NIST. American National Standard for Information Technology.

Sandhu, R., Ferraiolo, D., & Kuhn, R. (2000). *The NIST model for role-based access control: Towards a unified standard.* ACM. proceedings of 5th ACM Workshop on Role-Based Access Control.

Shu Gao, Y. D. (2002). *Applying Aspect-Orientation in Designing Security Systems: A Case Study.* School of Computer Science, Florida International University.

8.1 Referencias Bibliográficas

AspectJ, MDA, ITP/OCL, RBAC, Java, SecureUML, Design by Contract.

9 Apéndice I : Modelo completo de la política de seguridad aplicada a TRC. Entrada al Transformador.

Archivo TRC

```
createSecurityDiagram("TRC") ;

insertRole("TestOperator") ;

insertRole("TestSupervisor") ;

insertRole("TestAdministrator") ;

insertRoleHierarchy( "TestSupervisor", "TestOperator") ;

insertRoleHierarchy( "TestAdministrator", "TestSupervisor") ;

insertEntity( "Trc","Trc") ;

insertAttribute( "Trc", "owner","owner","String") ;

insertAttribute( "Trc", "scope","scope","TypeOfScope") ;

insertAttribute( "Trc", "name","name","String") ;

insertAttribute( "Trc", "description","description","String") ;

insertAttribute( "Trc", "temporal","temporal","boolean") ;

insertAttribute( "Trc", "hasTemporal","hasTemporal","boolean") ;

insertAttribute( "Trc", "initial","initial","boolean") ;

insertAttribute( "Trc", "TES","TES","String") ;

insertMethod( "Trc", "read","read","void"," ", true) ;

insertMethod( "Trc", "create","create","void","String pOwner,String pName,String
pDescr,TypeOfScope pScope,String pTes", true) ;

insertMethod( "Trc", "delete","delete","void"," ", true) ;

insertMethod( "Trc", "edit","edit","void","String pName, String pDescr, boolean pInCreation", true) ;

insertMethod( "Trc", "addTROSI","addTROSI","void","TROS pTros", true) ;

insertMethod( "Trc", "editTROSI","editTROSI","void","TROS pTros", true) ;

insertMethod( "Trc", "deleteTROSI","deleteTROSI","void","TROS pTros", true) ;

insertMethod( "Trc", "duplicate","duplicate","Trc","String pOwner,String pName,String
pDescr,TypeOfScope pScope,Set<TROSI> pSetTrosi", true) ;

insertMethod( "Trc","makeTemporal","makeTemporal","void","String pOwner,String pName", true) ;

insertMethod( "Trc", saveTemporal","saveTemporal","void","String pName,String pDescr,Set<TROSI>
pSetTrosi", true) ;
```

```
insertMethod( "Trc","discardTemporal","discardTemporal","void"," ", true) ;

insertAuthConstraint( "NewTRCPrivateAC", mkValue("Java"), mkValue("pOwner.equals( caller.name )
&& pScope == TypeOfScope.Private")) ;

insertAuthConstraint( "NewTRCGlobalAC", mkValue("Java"), mkValue("pOwner.equals(caller.name)"))
;

insertPermission( "NewTRCGlobal") ;

insertPermission( "NewTRCPrivate") ;

insertPermissionAssignment( "NewTRCGlobal", "TestSupervisor") ;

insertPermissionAssignment( "NewTRCPrivate", "TestOperator") ;

insertConstraintAssignment( "NewTRCGlobalAC", "NewTRCGlobal") ;

insertConstraintAssignment( "NewTRCPrivateAC", "NewTRCPrivate") ;

grantAtomicExecuteAccess( "create", "NewTRCGlobal") ;

grantAtomicExecuteAccess( "create", "NewTRCPrivate") ;

insertAuthConstraint( "MakeTemporalAC", mkValue("Java"),mkValue("true")) ;

insertPermission( "MakeTemporal") ;

insertPermissionAssignment( "MakeTemporal", "TestOperator") ;

insertConstraintAssignment( "MakeTemporalAC", "MakeTemporal") ;

grantAtomicExecuteAccess( "makeTemporal", "MakeTemporal");

insertAuthConstraint( "SaveTemporalOwnerAC", mkValue("Java"),
mkValue("self.owner.equals(caller.name)")) ;

insertPermission( "SaveTemporalOwner") ;

insertPermissionAssignment( "SaveTemporalOwner", "TestOperator") ;

insertConstraintAssignment( "SaveTemporalOwnerAC", "SaveTemporalOwner") ;

grantAtomicExecuteAccess( "saveTemporal", "SaveTemporalOwner");

insertAuthConstraint( "SaveTemporalAdminAC", mkValue("Java"), mkValue("true")) ;

insertPermission( "SaveTemporalAdmin") ;

insertPermissionAssignment( "SaveTemporalAdmin", "TestAdministrator") ;

insertConstraintAssignment( "SaveTemporalAdminAC", "SaveTemporalAdmin") ;

grantAtomicExecuteAccess( "saveTemporal", "SaveTemporalAdmin");

insertAuthConstraint( "DiscardTemporalAC", mkValue("Java"), mkValue("true")) ;

insertPermission( "DiscardTemporal") ;

insertPermissionAssignment( "DiscardTemporal", "TestOperator") ;

insertConstraintAssignment( "DiscardTemporalAC", "DiscardTemporal") ;
```

```
grantAtomicExecuteAccess( "discardTemporal", "DiscardTemporal");

insertAuthConstraint( "ReadTRCGlobalAC", mkValue("Java"), mkValue("self.scope ==
TypeOfScope.Global")) ;

insertPermission( "ReadTRCGlobal") ;

insertPermissionAssignment( "ReadTRCGlobal", "TestOperator") ;

insertConstraintAssignment( "ReadTRCGlobalAC", "ReadTRCGlobal") ;

grantEntityReadAccess( "Trc", "ReadTRCGlobal");

insertAuthConstraint( "ReadTRCOwnerAC", mkValue("Java"), mkValue("self.scope ==
TypeOfScope.Private && self.owner.equals ( caller.name )")) ;

insertPermission( "ReadTRCOwner") ;

insertPermissionAssignment( "ReadTRCOwner", "TestOperator") ;

insertConstraintAssignment( "ReadTRCOwnerAC", "ReadTRCOwner") ;

grantEntityReadAccess( "Trc", "ReadTRCOwner");

insertAuthConstraint( "ReadTRCAdminAC", mkValue("Java"), mkValue("true")) ;

insertPermission( "ReadTRCAdmin") ;

insertPermissionAssignment( "ReadTRCAdmin", "TestSupervisor") ;

insertConstraintAssignment( "ReadTRCAdminAC", "ReadTRCAdmin") ;

grantEntityReadAccess( "Trc", "ReadTRCAdmin");

insertAuthConstraint( "DuplicateTRCOperatorAC", mkValue("Java"),
mkValue("(self.scope==TypeOfScope.Global || (self.scope==TypeOfScope.Private &&
self.owner.equals( caller.name )))" +"&& (pScope==TypeOfScope.Private &&
caller.name.equals(pOwner))")) ;

insertPermission( "DuplicateTRCOperator") ;

insertPermissionAssignment( "DuplicateTRCOperator", "TestOperator") ;

insertConstraintAssignment( "DuplicateTRCOperatorAC", "DuplicateTRCOperator") ;

grantAtomicExecuteAccess( "duplicate", "DuplicateTRCOperator");

insertAuthConstraint( "DuplicateTRCGlobalAC", mkValue("Java"), mkValue("caller.name.equals(
pOwner )")) ;

insertPermission( "DuplicateTRCGlobal") ;

insertPermissionAssignment( "DuplicateTRCGlobal", "TestSupervisor") ;

insertConstraintAssignment( "DuplicateTRCGlobalAC", "DuplicateTRCGlobal") ;

grantAtomicExecuteAccess( "duplicate", "DuplicateTRCGlobal");

insertAuthConstraint( "EditTRCOwnerAC", mkValue("Java"), mkValue("caller.name.equals( self.owner
)")) ;
```

```
insertPermission( "EditTRCOwner") ;

insertPermissionAssignment( "EditTRCOwner", "TestOperator") ;

insertConstraintAssignment( "EditTRCOwnerAC", "EditTRCOwner") ;

grantAtomicExecuteAccess( "edit", "EditTRCOwner");

grantAtomicExecuteAccess( "addTROSI", "EditTRCOwner");

grantAtomicExecuteAccess( "deleteTROSI", "EditTRCOwner");

grantAtomicExecuteAccess( "editTROSI", "EditTRCOwner");

insertAuthConstraint( "EditTRCAdminAC", mkValue("Java"), mkValue("true")) ;

insertPermission( "EditTRCAdmin") ;

insertPermissionAssignment( "EditTRCAdmin", "TestAdministrator") ;

insertConstraintAssignment( "EditTRCAdminAC", "EditTRCAdmin") ;

grantAtomicExecuteAccess( "edit", "EditTRCAdmin");

grantAtomicExecuteAccess( "addTROSI", "EditTRCAdmin");

grantAtomicExecuteAccess( "deleteTROSI", "EditTRCAdmin");

grantAtomicExecuteAccess( "editTROSI", "EditTRCAdmin");

insertAuthConstraint( "EditTRCTemporalAC", mkValue("Java"), mkValue("self.temporal==true")) ;

insertPermission( "EditTRCTemporal") ;

insertPermissionAssignment( "EditTRCTemporal", "TestOperator") ;

insertConstraintAssignment( "EditTRCTemporalAC", "EditTRCTemporal") ;

grantAtomicExecuteAccess( "edit", "EditTRCTemporal");

insertAuthConstraint( "DeleteTRCOwnerAC", mkValue("Java"), mkValue("self.owner.equals(
caller.name )")) ;

insertPermission( "DeleteTRCOwner") ;

insertPermissionAssignment( "DeleteTRCOwner", "TestOperator") ;

insertConstraintAssignment( "DeleteTRCOwnerAC", "DeleteTRCOwner") ;

grantAtomicExecuteAccess( "delete", "DeleteTRCOwner");

insertAuthConstraint( "DeleteTRCAdminAC", mkValue("Java"), mkValue("true")) ;

insertPermission( "DeleteTRCAdmin") ;

insertPermissionAssignment( "DeleteTRCAdmin", "TestAdministrator") ;

insertConstraintAssignment( "DeleteTRCAdminAC", "DeleteTRCAdmin") ;

grantAtomicExecuteAccess( "delete", "DeleteTRCAdmin");
```

10 Apéndice II: Salida del Transformador para la política de seguridad aplicada a TRC.

Clase Trc.java, Interfaz Env.java y Aspecto TrcAC_Aspect.aj

Trc.java, es la clase abstracta que representa a Trc:

```
package TRC;

import java.util.*;


public abstract class Trc {

// attributes

protected String owner;

protected TypeOfScope scope;

protected String name;

protected String description;

protected boolean temporal;

protected boolean hasTemporal;

protected boolean initial;

protected String TES;


// methods

public String getowner() throws Exception {return owner; }

public void setowner(String owner) throws Exception { this.owner = owner; }

public TypeOfScope getscope() throws Exception {return scope; }

public void setscope(TypeOfScope scope) throws Exception { this.scope = scope; }

public String getname() throws Exception {return name; }

public void setname(String name) throws Exception { this.name = name; }

public String getdescription() throws Exception {return description; }

public void setdescription(String description) throws Exception { this.description =
description; }

public boolean gettemporal() throws Exception {return temporal; }

public void settemporal(boolean temporal) throws Exception { this.temporal = temporal;
}

public boolean gethasTemporal() throws Exception {return hasTemporal; }

public void sethasTemporal(boolean hasTemporal) throws Exception { this.hasTemporal =
hasTemporal; }

public boolean getinitial() throws Exception {return initial; }
```

```
public void setinitial(boolean initial) throws Exception { this.initial = initial; }

public String getTES() throws Exception {return TES; }

public void setTES(String TES) throws Exception { this.TES = TES; }

public abstract void read ( ) throws Exception;

public abstract void create (String pOwner) throws Exception;

public abstract void delete ( ) throws Exception;

public abstract void edit (String pName) throws Exception;

public abstract void addTROSI (TROS pTros) throws Exception;

public abstract void editTROSI (TROS pTros) throws Exception;

public abstract void deleteTROSI (TROS pTros) throws Exception;

public abstract Trc duplicate (String pOwner) throws Exception;

public abstract void makeTemporal (String pOwner) throws Exception;

public abstract void saveTemporal (String pName) throws Exception;

public abstract void discardTemporal ( ) throws Exception;

}
```

Env.java, la interfaz para obtener el Role del usuario actual:

```
package Env;
import Role.*;

public class Env {
        public static Role getUsrRole(){
                //put here your implementation to obtain the role of the user.
        return null;
        }
}
```

Trc_ACAspect.aj, el aspecto de Access Control:

```
package TRC;

import java.util.*;

import Role.*;

import Env.*;


public aspect Trc_ACAspect {


 private static Role caller ;


 // PointCuts


pointcut PCgetowner( Trc  self ): call(String Trc.getowner())&&target( self );

pointcut PCgetscope( Trc  self ): call(TypeOfScope Trc.getscope())&&target( self );

pointcut PCgetname( Trc  self ): call(String Trc.getname())&&target( self );
```

```
pointcut PCgetdescription( Trc  self ): call(String Trc.getdescription())&&target(
self );

pointcut PCgettemporal( Trc  self ): call(boolean Trc.gettemporal())&&target( self );

pointcut PCgethasTemporal( Trc  self ): call(boolean Trc.gethasTemporal())&&target(
self );

pointcut PCgetinitial( Trc  self ): call(boolean Trc.getinitial())&&target( self );

pointcut PCgetTES( Trc  self ): call(String Trc.getTES())&&target( self );

pointcut PCcreate( Trc  self , String pOwner): call(void Trc.create( String
))&&target( self )&&args(pOwner);

pointcut PCdelete( Trc  self ): call(void Trc.delete( ))&&target( self );

pointcut PCedit( Trc  self , String pName): call(void Trc.edit( String ))&&target(
self )&&args(pName);

pointcut PCaddTROSI( Trc  self , TROS pTros): call(void Trc.addTROSI( TROS ))&&target(
self )&&args(pTros);

pointcut PCeditTROSI( Trc  self , TROS pTros): call(void Trc.editTROSI( TROS
))&&target( self )&&args(pTros);

pointcut PCdeleteTROSI( Trc  self , TROS pTros): call(void Trc.deleteTROSI( TROS
))&&target( self )&&args(pTros);

pointcut PCduplicate( Trc  self , String pOwner): call(Trc Trc.duplicate( String
))&&target( self )&&args(pOwner);

pointcut PCmakeTemporal( Trc  self , String pOwner): call(void Trc.makeTemporal(
String ))&&target( self )&&args(pOwner);

pointcut PCsaveTemporal( Trc  self , String pName): call(void Trc.saveTemporal( String
))&&target( self )&&args(pName);

pointcut PCdiscardTemporal( Trc  self ): call(void Trc.discardTemporal( ))&&target(
self );




// PointCuts Before Advices



before ( Trc  self  ) throws Exception:PCgetowner(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }
```

```
        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgetscope(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                         else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgetname(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){
```

```
                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgetdescription(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }
```

```
        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgettemporal(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgethasTemporal(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);
```

```
        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgetinitial(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                         else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before ( Trc  self  ) throws Exception:PCgetTES(  self  ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){
```

```
                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.scope == TypeOfScope.Private && self.owner.equals (
caller.name ) ) return ;

                if ( self.scope == TypeOfScope.Global ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , String pOwner) throws Exception:PCcreate( self , pOwner){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( pOwner.equals(caller.name) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( pOwner.equals( caller.name ) && pScope == TypeOfScope.Private )
return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);
```

```
}


before (Trc  self ) throws Exception:PCdelete( self ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.owner.equals( caller.name ) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , String pName) throws Exception:PCedit( self , pName){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( caller.name.equals( self.owner ) ) return ;
```

```
                if ( self.temporal==true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , TROS pTros) throws Exception:PCaddTROSI( self , pTros){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                         else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( caller.name.equals( self.owner ) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , TROS pTros) throws Exception:PCeditTROSI( self , pTros){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                         else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;
```

```
                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( caller.name.equals( self.owner ) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , TROS pTros) throws Exception:PCdeleteTROSI( self , pTros){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                         else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( caller.name.equals( self.owner ) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , String pOwner) throws Exception:PCduplicate( self , pOwner){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){
```

```
                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestSupervisor){

                if ( caller.name.equals( pOwner ) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( (self.scope==TypeOfScope.Global ||
(self.scope==TypeOfScope.Private && self.owner.equals( caller.name ))) +&&
(pScope==TypeOfScope.Private && caller.name.equals(pOwner)) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , String pOwner) throws Exception:PCmakeTemporal( self , pOwner){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestOperator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self , String pName) throws Exception:PCsaveTemporal( self , pName){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){
```

```
                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestAdministrator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        if ( caller instanceof TestOperator){

                if ( self.owner.equals(caller.name) ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}



before (Trc  self ) throws Exception:PCdiscardTemporal( self ){

        caller=Env.getUsrRole();

        String argumentsInError="";

        for (int i=0;i<thisJoinPoint.getArgs().length;i++){

                if
(thisJoinPoint.getArgs()[i]!=null)argumentsInError+="\n\t\t"+thisJoinPoint.getArgs()[i
].toString();

                        else argumentsInError+="\n\t\tnull;";       }

        if ( caller instanceof TestOperator){

                if ( true ) return ;

                        else throw new Exception("Permission error: Role:
"+caller.getClass().getName()+"\n In: "+thisJoinPoint+"\n Attributes:
"+argumentsInError);

        }

        throw new Exception("Role error: Role: "+caller.getClass().getName()+"\n In:
"+thisJoinPoint+"\n Attributes: "+argumentsInError);

}




}
```

11 Apéndice III: Interfaces de los componentes de la implementación del transformador.

API OCLEval:

A través de la API de OCLEval, se puede crear un proceso Maude sobre el que:

- Cargamos ITP/OCL.

```
void loadMaude() throws Exception
```

- Cerrar el proceso Maude.

```
public void exit()
```

- Creamos un diagrama de clases.

```
public String createClassDiagram(String diagram) throws Exception
```

- Introducimos el diagrama de clases el metamodelo de SecureUML a través de:

```
public String createClassDiagram(String diagram) throws Exception

public String insertClass(String diagram, String classname) throws Exception

public String insertSubclass(String diagram, String subclassname, String superclassname) throws Exception

public String insertAssociation(String diagram, String classname1, String rolename1, String mult1, String mult2, String rolename2, String classname2)

public String insertAttribute(String diagram, String classname,
   String attrname, String attrtype) throws Exception
```

- Insertamos operaciones OCL en el metamodelo.

```
public String insertParameterlessOperationWithoutContext(String
   classdiagram, String operationname, String returntype,
   String body)

public String insertParameterlessOperationWithoutContext(String
   classdiagram, String operationname, String returntype,
   String body)

public String insertParameterizedOperation(String classdiagram,
   String context, String operationname, String parametername,
   String parametertype, String returntype, String body)
```

- Introducimos invariantes sobre metamodelo y comprobarlos.

```
public String insertContextualizedInvariant(String diagram,
String classname, String invariant) throws Exception
```

```
public String insertGeneralInvariant(String diagram, String
invariant) throws Exception

public String checkInvariants(String classdiagram, String
objectdiagram) throws Exception
```

- Creamos un diagrama de objetos.

```
public String createObjectDiagram(String classdiagram, String
objectdiagram) throws Exception
```

- Introducimos el diagrama de objetos una instancia particular de SecureUML a través de:

```
public String insertObject(String objectdiagram, String
classname, String objectname) throws Exception

public String insertLink(String objectdiagram, String
classname1, String rolename1, String rolename2, String
classname2, String object1, String object2)

public String deleteLink(String objectdiagram, String
classname1, String rolename1, String rolename2, String
classname2, String object1, String object2)

public String insertAttributeValue(String objectdiagram, String
classname, String object, String attr, String attrType, String
value)
```

- Realizamos preguntas sobre el diagrama de objetos.

```
public String query(String classdiagram, String objectdiagram,
String oclexp)
```

API SUMLEval:

- Crear un diagrama de seguridad. Dicho método envía el mensaje a OCLEval para que construya el diagrama de objetos y inserta ya en él todos los objetos que tiene que haber por defecto, tal y como deben existir en SecureUML:

```
public String createSecurityDiagram(String diagram) throws
Exception
```

- API para crear el diagrama de objetos SecureUML en ITP/OCL:

```
public String insertRole(String diagram, String role) throws
Exception

public String insertRoleHierarchy(String diagram, String
subrole, String superrole) throws Exception


public String insertUser(String diagram, String user) throws
Exception


public String insertUserAssignment(String diagram, String user,
String role) throws Exception
```

```
public String insertEntity(String diagram, String entity,
String name) throws Exception

public String insertAttribute(String diagram, String entity, S
String attribute, String name, String type) throws Exception

public String insertMethod(String diagram, String entity,
String method,String name,String type,String parameters,
Boolean isquery) throws Exception

public String insertAuthConstraint(String diagram, String auth,
String lang, String constr) throws Exception

public String insertAssociationEnd(String diagram, String
entity, String assocend) throws Exception

public String insertPermission(String diagram, String
permission) throws Exception
```

- Estos dos métodos son importantes ya que con ellos asignamos o permisos con roles o authorization constraints con permisos.

```
public String insertPermissionAssignment(String diagram, String
permission, String role) throws Exception

public String insertConstraintAssignment(String diagram, String
auth, String permission) throws Exception
```

- Con los métodos que mostramos a continuación se asignan permisos según sea su Action. Los propios métodos borran el permiso por defecto que se había creado y asignado anteriormente al introducir el objeto, posteriormente crean un nuevo enlace con el permiso que entra como parámetro, mostramos el código para grantAtomicExecuteAccess:

```
public String grantEntityFullAccess(String diagram, String
entity, String permission) throws Exception

public String grantEntityReadAccess(String diagram, String
entity, String permission) throws Exception

public String grantEntityUpdateAccess(String diagram, String
entity, String permission) throws Exception

public String grantAtomicCreateAccess(String diagram, String
entity, String permission) throws Exception
```

```
public String grantAtomicDeleteAccess(String diagram, String
entity, String permission) throws Exception


public String grantAttributeFullAccess(String diagram, String
attribute, String permission) throws Exception


public String grantAtomicUpdateAccess(String diagram, String
attribute, String permission) throws Exception


public String grantAtomicReadAccess(String diagram, String
attribute, String permission) throws Exception


public String grantAtomicExecuteAccess(String diagram, String
method, String permission) throws Exception
```

- API para hacer querys OCL al diagrama de objetos:

```
public List getEntityAttributes(String diagram, String entity)
throws Exception


public List getEntityMethods(String diagram, String entity)
throws Exception


public List getEntityQueryMethods(String diagram, String
entity) throws Exception


public List getEntityNonQueryMethods(String diagram, String
entity) throws Exception


public List getEntities(String diagram) throws Exception

public String getEntityName(String diagram,String entity)
throws Exception


public String getAttributeName(String diagram,String attribute)
throws Exception


public String getAttributeType(String diagram,String attribute)
throws Exception


public String getMethodName(String diagram,String method)
throws Exception


public String getRoleName(String diagram,String auth) throws
Exception


public String getNameEntityWhoHasMethod(String diagram,String
method) throws Exception
```

```
public String getNameEntityWhoHasAttribute(String
diagram,String attribute) throws Exception


public String getMethodType(String diagram,String method)
throws Exception


public String getMethodParameters(String diagram,String method)
throws Exception


public String getMetodParametersTypes(String diagram,String
method) throws Exception


public String getParameterWithoutTypes(String diagram,String
method) throws Exception


public Boolean hasHigherRole(String diagram,String
Constraint1,String Constraint2) throws Exception


public List getAttributeReadConstraints(String diagram,String
attr) throws Exception


public List getAttributeWriteConstraints(String diagram,String
attr) throws Exception


public List getMethodConstraints(String diagram,String method)
throws Exception


public String getAuthorizationBody(String diagram,String
constraint) throws Exception
```

12 Apéndice IV : Casos de Prueba de TRC Manager

Primero vamos a hacer que no se pueda crear un TRC porque no cumplimos las restricciones sobre el permiso NewTRCPrivate si el Scope es Global:

Realizamos un login con:

User: Jorge

Role: TestOperator

Se observa en Figure 12-1

Figure 12-1

Creamos un Trc con Scope Global Figure 12-2

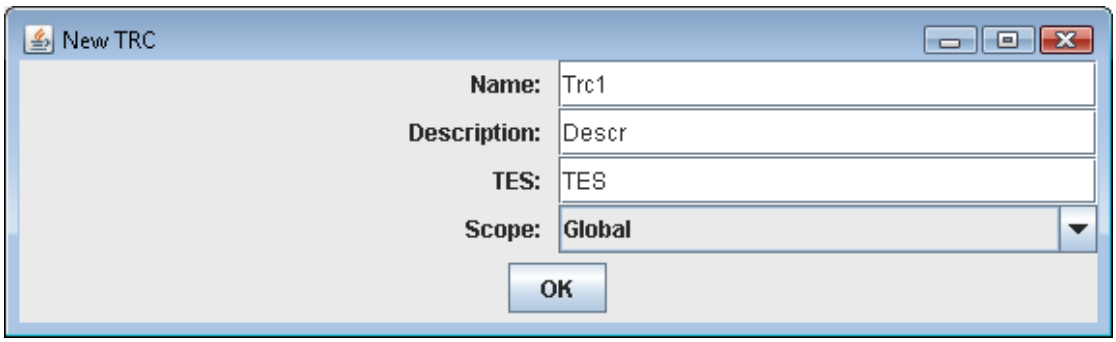

Figure 12-2

Se muestra en error Figure 12-3

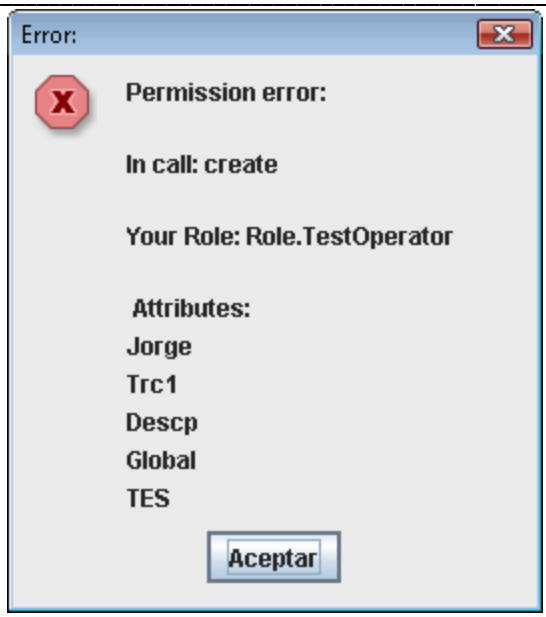

Figure 12-3

Ahora si ponemos Scope como Private si cumplimos las restricciones y si se puede crear el TRC. Se muestra en las Figure 12-4 y Figure 12-5

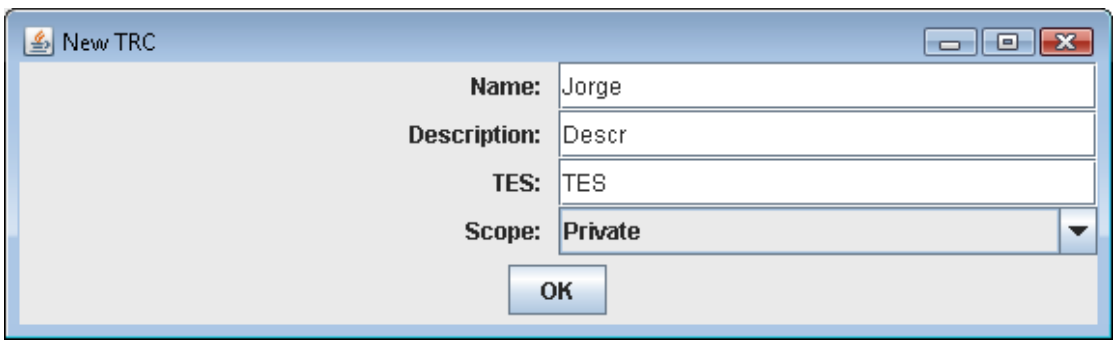

Figure 12-4

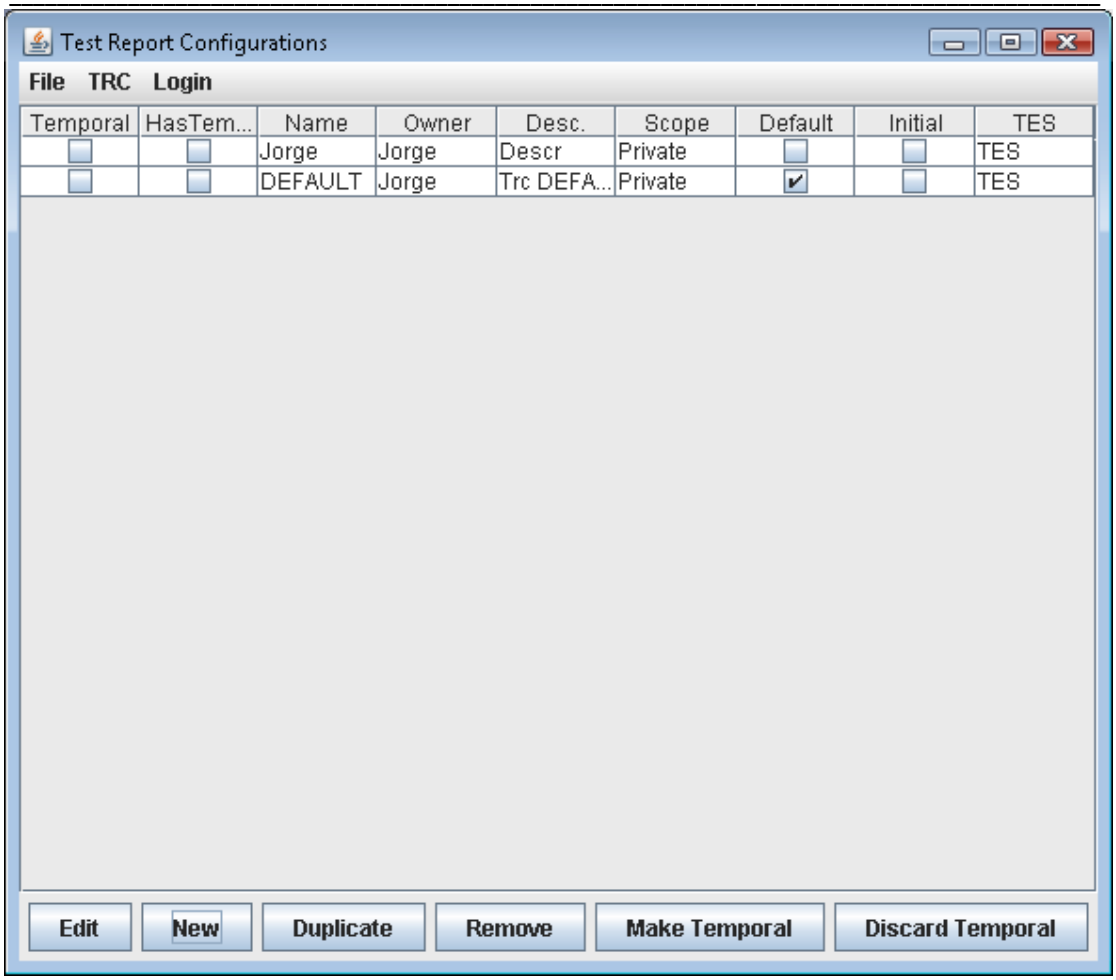

Figure 12-5

Vamos a cambiar de usuario y Role, realizamos un nuevo login, se observa en Figure 12-6

User: Pedro

Role: Any

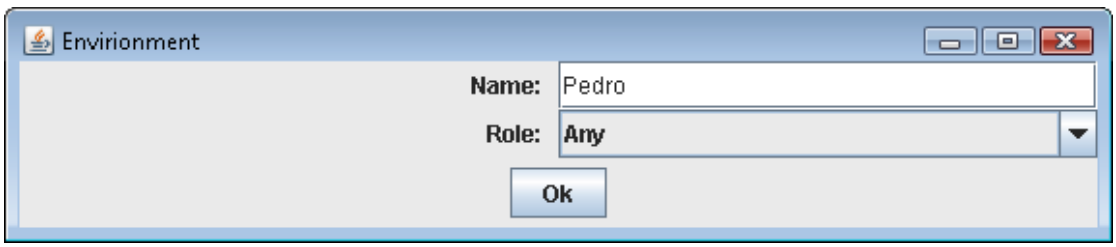

Figure 12-6

Pedro tiene un role por defecto que no tiene permisos. Cualquier acción va a generar un error de Role. Al hacer login se intenta crear el Trc por defecto y como se genera un error nunca un usuario con el Role Any podrá entrar en la aplicación. Ver la Figure 12-7

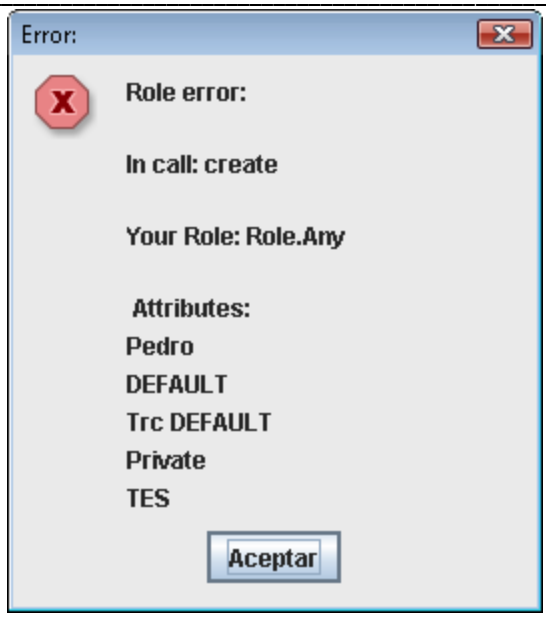

Figure 12-7

Realizamos un login con:

User: Christiano

Role: TestAdministrator

Hemos omitido la captura de pantalla de Login. Mostramos un error que deriva de las restricciones de la aplicación. El Trc por defecto nunca puede ser borrado. Cuando lo intentamos aparece el siguiente error Figure 12-8

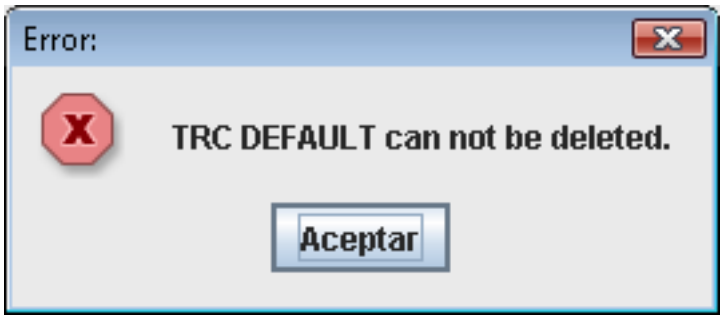

Figure 12-8

Al ser Christiano TestAdministrator, el Role con mas privilegios, podrá hacer que el Trc DEFAULT pueda ser temporal y luego hacer que no sea. La Figure 12-9 se origina tras hacer click en MakeTeporal. La Figure 12-10 se origina tras hacer click en DiscardTemporal.

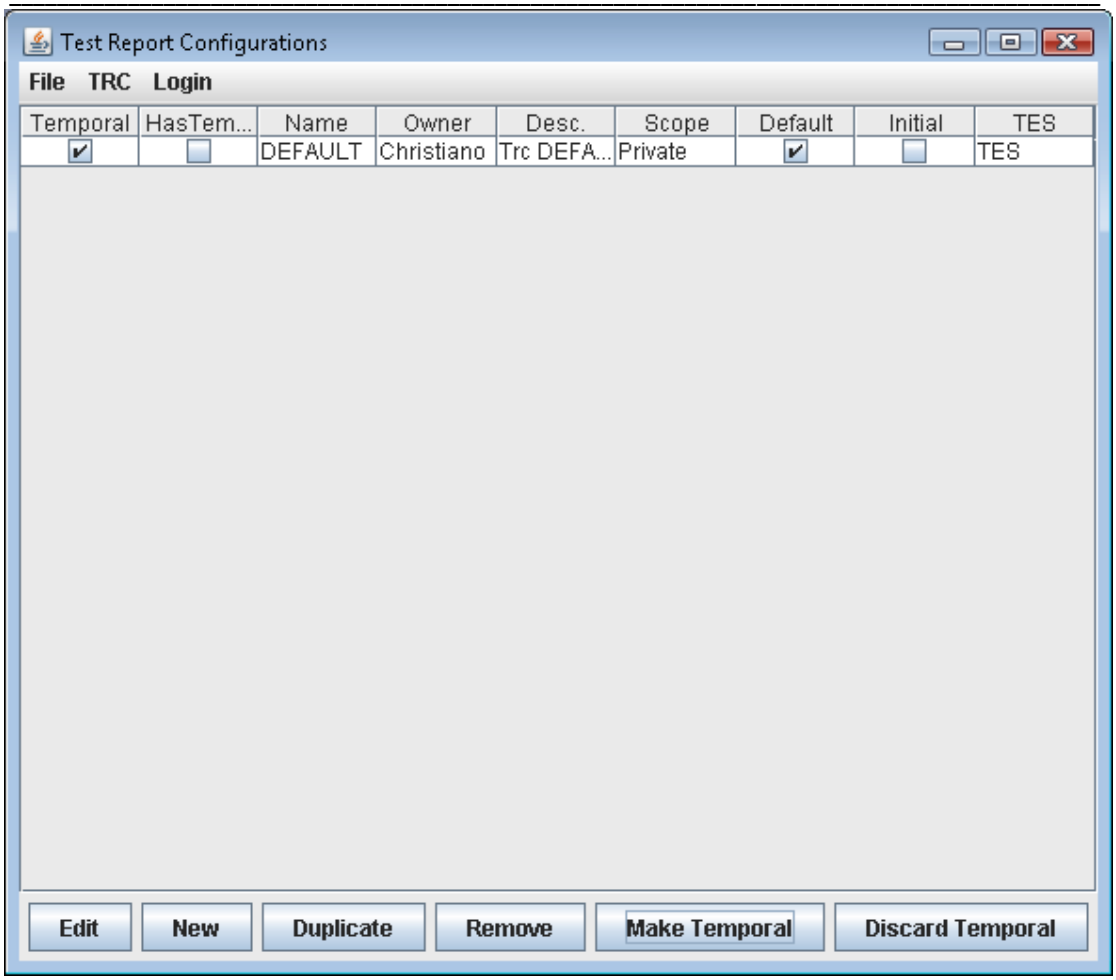
Test Report Configurations
File
TRC
Login
Temporal
HasTem...
Name
Owner
Desc.
Scope
Default
Initial
TES
DEFAULT
Christiano
Trc DEFA...
Private
TES
Edit
New
Duplicate
Remove
Make Temporal
Discard Temporal

Figure 12-9

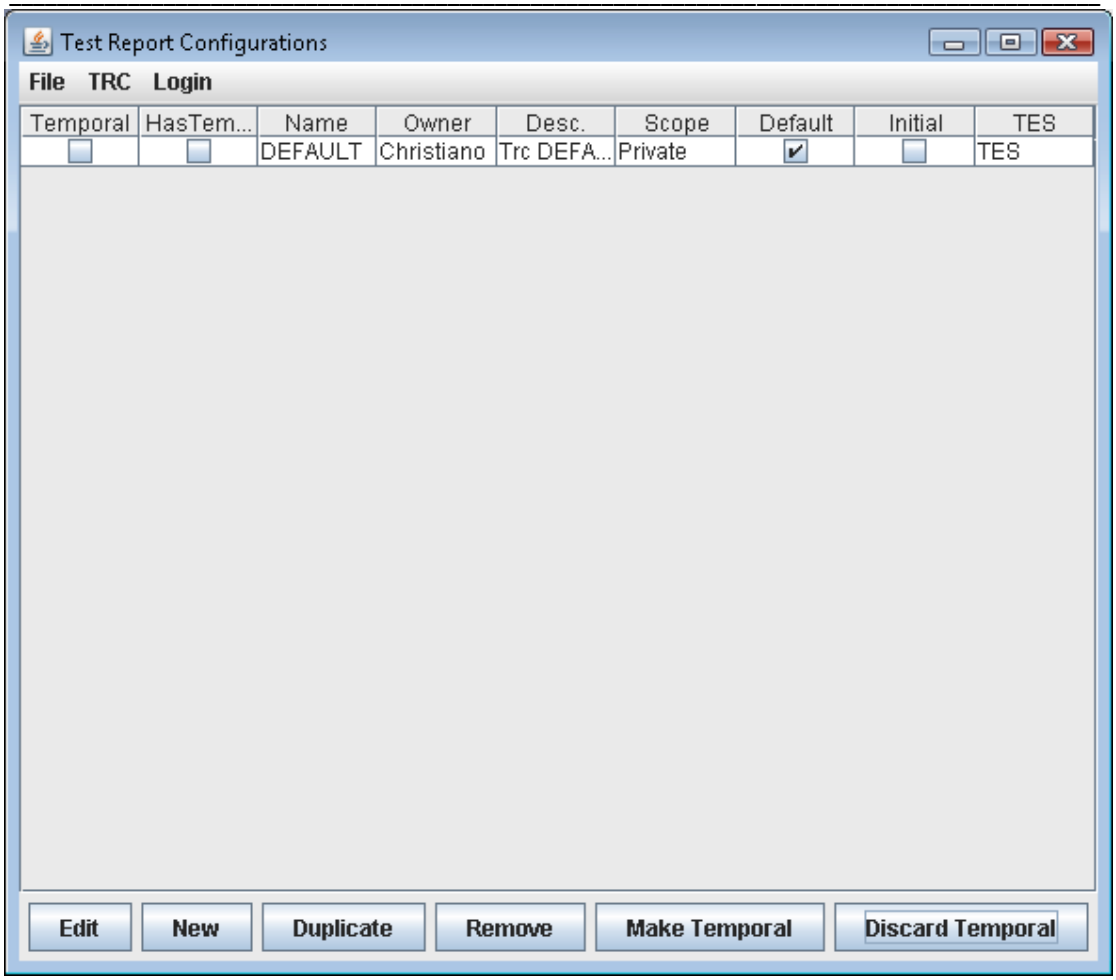

Figure 12-10

Dejamos al lector que pruebe los casos para Remove y Edit para no hacer demasiado extensa la captura de resultado de diferentes Test.